

Accessibility for People with Disabilities

Microsoft is committed to making its products and services easier for everyone to use. This appendix provides information about the following features, products and services, which make Microsoft Windows, Microsoft Windows NT, and Microsoft Visual J++ more accessible for people with disabilities:

- Microsoft services for people who are deaf or hard-of-hearing
- Access Packs for either Microsoft Windows or Microsoft Windows NT, a software utility that makes using Windows or Windows NT easier for people with motion or hearing disabilities
- Keyboard layouts designed for people who type with one hand or a wand
- Microsoft software documentation on audio cassette, floppy disk, or compact disc (CD)
- Third-party utilities to enhance accessibility
- Hints for customizing Microsoft Windows or Microsoft Windows NT
- Other products and services for people with disabilities

Note The information in this section applies only to users who purchased Microsoft products in the United States. If you purchased Windows or Windows NT outside the United States, your package contains a subsidiary information card listing Microsoft support services telephone numbers and addresses. You can contact your subsidiary to find out whether the type of products and services described in this topic are available in your area.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Microsoft Services for People Who Are Deaf or Hard-of-Hearing

Through a text telephone (TT/TDD) service, Microsoft provides people who are deaf or hard-of-hearing with complete access to Microsoft product and customer services.

You can contact Microsoft Sales Information Center on a text telephone by dialing (800) 892-5234 between 6:30 A.M. and 5:30 P.M. Pacific time. For technical assistance in the United States, you can contact Microsoft Support Network on a text telephone at (206) 635-4948 between 6:00 A.M. and 6:00 P.M. Pacific time, Monday through Friday, excluding holidays. In Canada, dial (905) 568-9641 between 8:00 A.M. and 8:00 P.M. Eastern time, Monday through Friday, excluding holidays. Microsoft support services are subject to Microsoft prices, terms, and conditions in place at the time the service is used.

{ewl msdn.cd.dll, ewcright, /c"Microsoft"}

Access Packs for Microsoft Windows and Microsoft Windows NT

Microsoft distributes Access Packs for Microsoft Windows and Microsoft Windows NT, which provide people with motion or hearing disabilities better access to computers running Windows or Windows NT. (If you are running Microsoft Windows 95, these same Access Pack features are already built in. See online Help for more information.) Microsoft Windows and Microsoft Windows NT Access Packs contain several features that:

- Allow single-finger typing of SHIFT, CTRL, and ALT key combinations.
- Ignore accidental keystrokes.
- Adjust the rate at which a character is repeated when you hold down a key, or turn off character repeating entirely.
- Prevent extra characters if you unintentionally press a key more than once.
- Enable you to control the mouse cursor by using the keyboard.
- Enable you to control the computer keyboard and mouse by using an alternate input device.
- Provide a visual cue when the computer beeps or makes other sounds.

Access Pack for Microsoft Windows is included on the Microsoft Windows Driver Library in the file ACCP.EXE. Access Pack for Microsoft Windows NT is included in the Microsoft Application Note WNO789. If you have a modem, you can download ACCP.EXE or WNO789.EXE, which are self-extracting archive files, from the following network services:

- CompuServe®
- GENie™
- The Microsoft Network
- Microsoft Download Service (MSDL), which you can reach by calling (206) 936-6735 any time except between 1:00 A.M. and 2:30 A.M. Pacific time. Use the following communications settings:

	For this setting	Specify
	Baud rate	1200, 2400, 9600, or 14400
	Parity	None
	Data bits	8
	Stop bits	1
•	Various user-group bulletin boards (such as the bulletin-board services on the Association of PC User Groups network)	
•	In /SOFTLIB/MSLFILES on the Internet servers FTP.MICROSOFT.COM and WWW.MICROSOFT.COM	

People within the United States who do not have a modem can order the Access Packs on disks by calling Microsoft Sales Information Center at (800) 426-9400 (voice) or (800) 892-5234 (text telephone). In Canada, you can call (905) 568-3503 or (905) 568-9641 (text telephone).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Keyboard Layouts for Single-Handed Users

Microsoft distributes Dvorak keyboard layouts that make the most frequently typed characters on a keyboard more accessible to people who have difficulty using the standard "QWERTY" layout. There are three Dvorak layouts: one for two-handed users, one for people who type with their left hand only, and one for people who type with their right hand only. The left-handed or right-handed keyboard layouts can also be used by people who type with a single finger or a wand. You do not need to purchase any special equipment to use these features.

Microsoft Windows and Microsoft Windows NT already support the two-handed Dvorak layout, which can be useful for coping with or avoiding types of repetitive-motion injuries associated with typing. To get this layout use the Windows Control Panel; consult your on-line documentation for detailed instructions. The two layouts for people who type with one hand are distributed as Microsoft Application Note GA0650. This application note is also contained in file GA0650.EXE on most network services and on the Microsoft Download Service. For instructions on obtaining this application note, see [Access Packs for Microsoft Windows and Microsoft Windows NT](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Microsoft Documentation in Alternative Formats

People who have difficulty reading or handling printed documentation may obtain many Microsoft publications from Recording for the Blind, Inc. Recording for the Blind distributes these documents to registered, eligible members of their distribution service, either on audio cassettes or on floppy disks. The Recording for the Blind collection contains more than 80,000 titles, including Microsoft product documentation and books from Microsoft Press. You can contact Recording for the Blind at the following address or phone numbers for information on eligibility and availability of Microsoft product documentation and books from Microsoft Press:

Recordin	Phone:
g for the	(609)
Blind,	452-0606
Inc.	Fax:
20	(609)
Roszel	987-8116
Road	
Princeton	
, NJ	
08540	

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Third-Party Utilities to Enhance Accessibility

A wide variety of third-party hardware and software products are available to make personal computers easier to use for people with disabilities. Among the different types of products available for the MS-DOS, Microsoft Windows, and Microsoft Windows NT operating systems are:

- Programs that enlarge or alter the color of information on the screen for people with visual impairments.
- Programs that describe information on the screen in braille or synthesized speech for people who are blind or have difficulty reading.
- Hardware and software utilities that modify the behavior of the mouse and keyboard.
- Programs that enable users to “type” using a mouse or their voice.
- Word or phrase prediction software that allows one to type more quickly and with fewer keystrokes.
- Alternate input devices, such as single switch or puff-and-sip devices, for those who cannot use a mouse or a keyboard.

For more information on obtaining third-party utilities, see [Getting More Information for People with Disabilities](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Customizing Windows or Windows NT

There are many ways you can adjust the appearance and behavior of Windows or Windows NT to suit varying vision and motor abilities without requiring any additional software or hardware. These include ways to adjust the appearance as well as the behavior of the mouse and keyboard. The specific methods available depend on which operating system you are using. Application notes are available describing the specific methods available for each operating system.

See the appropriate application note for information related to customizing your operating system for people with disabilities. For information on obtaining application notes, see [Access Packs for Microsoft Windows and Microsoft Windows NT](#).

Operating system	Application note
Microsoft Windows 3.0	WW0786. TXT
Microsoft Windows 3.1	WW0787. TXT
Microsoft Windows for Workgroups 3.1	WG0788. TXT
Microsoft Windows NT 3.1 and 3.5	WN0789. EXE
Microsoft Windows 95	WN1062

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Getting More Information for People with Disabilities

For more information on Microsoft products and services for people with disabilities, contact:

Microso Voice (800)
ft Sales telep 426-
Informa hone: 9400
tion Text (800)
Center telep 892-
One hone: 5234
Microso Fax: (206)
ft Way 635-
Redmo 6100
nd, WA
98052-
6393

The Trace R&D Center at the University of Wisconsin–Madison produces a book and a compact disc that describe products that help people with disabilities use computers. The book, titled *Trace ResourceBook*, provides descriptions and photographs of about 2,000 products. The compact disc, titled *CO-NET CD*, provides a database of more than 18,000 products and other information for people with disabilities. It is issued twice a year. To obtain these directories, contact:

Trace Voice (608)
R&D telep 263-
Center hone: 2309
S-151 Text (608)
Waism telep 263-
an hone: 5408
Center Fax: (608)
1500 262-
Highlan 8848
d
Avenue
Madiso
n, WI
53705-
2280

For general information and recommendations on how computers can help specific people, you should consult a trained evaluator who can best match your needs with the available solutions. An assistive technology program in your area will provide referrals to programs and services that are available to you. To locate the assistive technology program nearest you, you can contact:

Nationa Voice (803)
l /text 777-
Informa telep 4435
tion hone: (803)
System Fax: 777-
Center 6058
for
Develo
pmental
Disabilit
ies

Benson
Building
Univers
ity of
South
Carolin
a
Columb
ia, SC
29208

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Installing Visual J++

{ewl msdncd, EWGraphic, car0a 0 /a "build.bmp"} To install Visual J++

- 1** Insert the Visual J++ disc into your CD-ROM drive.
The AutoPlay feature of Windows 95 or Windows NT 4.0 will start the Visual J++ Master Setup automatically. The Master Setup screen appears.
- 2** Choose the Install Visual J++ option.
- 3** Follow the instructions on the screen.
- 4** Launch the application by choosing the Microsoft Developer Studio icon in the Microsoft Visual J++ group in the Start menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a Simple Java Applet

Use this procedure to create a basic Java™ applet that can be run from a Web page.

{ewl msdncd, EWGraphic, car1a 0 /a "build.bmp"} To create a basic Java applet to be run on a Web page

- 1 From the File menu, choose New.
The New dialog box appears.
- 2 Select Text File and click OK.
A blank text editor window appears.
- 3 Type the following code into the text editor:

```
import java.awt.Graphics;

class Hello1 extends java.applet.Applet
{
    public void paint( Graphics g )
    {
        g.drawString( "Hello, World!", 50, 25 );
    }
}
```

- 4 From the File menu, choose Save.
The Save dialog box appears.
- 5 Type `Hello1.java` and click Save.
- 6 From the Build menu, choose Build Hello1.
A dialog box appears, asking if you want to create a default workspace.
- 7 Click Yes.
Visual J++ creates a default workspace and builds the applet.
- 8 From the Build menu, choose Execute.
The Information for Running Class dialog box appears.
- 9 Type `Hello1` and click OK.
The Java applet runs.

Use Applet Wizard to generate a basic Java applet as a starting point for your own applets. For more information on using Applet Wizard, see the *Test Drive* in Books Online.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a Simple Java Application

Use this procedure to create a standalone Java application (i.e., one that does not run from a Web page).

{ewl msdncd, EWGraphic, car2a 0 /a "build.bmp"} To create a standalone Java application

- 1** If you previously created a Java applet, close the existing project by choosing Close Workspace from the File menu.
- 2** From the File menu, choose New.
The New dialog box appears.
- 3** Select Text File and click OK.
A blank text editor window appears.
- 4** Type the following code into the text editor:

```
class Hello2
{
    public static void main( String args[] )
    {
        System.out.println( "Hello, World!" );
    }
}
```

- 5** From the File menu, choose Save.
The Save dialog box appears.
- 6** Type `Hello2.java` and click Save.
- 7** From the Build menu, choose Build Hello2.
A dialog box appears, asking if you want to create a default workspace.
- 8** Click Yes.
Visual J++ creates a default workspace and builds the application.
- 9** From the Build menu, choose Execute.
The Information for Running Class dialog box appears.
- 10** Type `Hello2`.
- 11** Select the radio button "Stand-alone interpreter."
- 12** Choose OK.
The Java application runs.

You can use Applet Wizard to generate a Java applet that can be run as an application. For more information on using Applet Wizard, see the *Test Drive* in Books Online.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Getting Help

In Visual J++, the integrated InfoViewer is the central place for accessing documentation.

{ewl msdncd, EWGraphic, car3a 0 /a "build.bmp"} To view the Visual J++ Books Online

- 1 Launch the application by choosing the Microsoft Developer Studio icon in the Microsoft Visual J++ group in the Start menu.
- 2 Click the InfoView tab in the Project Workspace to see Books Online.
- 3 Click the "+" symbols next to each book to see its contents.
- 4 Double-click a document icon to view the topic.
A separate window opens with the topic text.

{ewl msdncd, EWGraphic, car3a 1 /a "build.bmp"} To get Help quickly while working in the text editor

- 1 In the text editor, place the cursor on, or next to, a keyword in your code.
- 2 Press F1.
The topic appears, or a list of topics that relate to the keyword appears.

{ewl msdncd, EWGraphic, car3a 2 /a "build.bmp"} To get Help on build errors

- 1 In the Output window, place the cursor on the build error number.
- 2 Press F1.
A more detailed explanation of the error appears.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Where Do I Go From Here?

- The Visual J++ website at <http://www.microsoft.com/visualj> is your key source for product information, updates, and other Internet resources.
- Choose *Introducing Visual J++* from the Master Setup screen for an animated look at Visual J++ features and technology.
- The *Learn Java Now* book explains Java programming fundamentals.
- Books Online provides comprehensive information about using Visual J++.
- The Samples in Books Online offer real-world Java code using COM, RDO, and other key technologies.
- Web Favorites on the Help menu gives easy access to key websites like the Microsoft Web and JavaSoft.
- The Usenet newsgroup `microsoft.public.visualj` offers discussions about Visual J++.
- If you have the Professional Edition of Visual J++, the Cool Tools directory on your CD contains third-party tools, wizards, class libraries, and controls that work with Visual J++.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Java and COM Together

Visual J++ is the first compiler that lets you take full advantage of Java Support in Internet Explorer. Internet Explorer provides the reference implementation of the Java virtual machine (or VM) for Win32 operating systems. This VM supports integration between Java and the Component Object Model (COM), the protocol underlying the ActiveX platform, Automation, and Object Linking and Embedding (OLE).

While the Java Support in Internet Explorer by itself provides benefits for any Java applet, you must use a development tool that supports this integration in order to take full advantage of both Java and COM.

Using Visual J++, you can give your Java program access to any of the software components that support COM, including thousands of ActiveX controls and Automation objects.

This document describes the following scenarios for using Java and COM together:

- Controlling a Java Applet through Scripting
- Using a COM object from Java
- Connecting a Java Applet and an OLE Control
- Exposing a Java Class as a COM Class

For background information on COM, see the Win32 Software Development Kit (SDK). For more information on developing for the ActiveX platform, see the ActiveX Software Development Kit. You can download the ActiveX SDK from <http://www.microsoft.com/activex/> (which is accessible using the Web Favorites command on the Help menu).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Controlling a Java Applet through Scripting

Web developers can embed ActiveX controls into their Web pages and drive them using Visual Basic Scripting Edition (VBScript). The ActiveX runtime for Java offers developers another option by making Java applets scriptable too. When a Java applet is running in a browser such as Internet Explorer, all the public methods and variables of the applet automatically become available to VBScript or any other language supporting the ActiveX scripting protocol.

As an example, consider a Java applet like the following:

```
class Buzzer extends Applet
{
    public int pitch;
    public void buzz()
    {
        //play a sound at the specified pitch
    }
    // other methods
    // ...
}
```

To include such an applet in your HTML page, simply use the <APPLET> tag, specifying the ID attribute so that you can refer to the applet from the scripting language:

```
<APPLET CODE="Buzzer.class" ID=doorbell>
```

You can then define buttons that allow the user to control the applet. For example:

```
<INPUT TYPE=button VALUE="Higher" NAME="BtnHigher">
<INPUT TYPE=button VALUE="Lower" NAME="BtnLower">
<INPUT TYPE=button VALUE="Play" NAME="BtnPlay">
```

The NAME attribute lets you refer to each button from the scripting language.

In the scripting portion of your HTML page, you can define `OnClick` handlers for each button, making them manipulate the applet:

```
<SCRIPT language = "VBScript">
<!--
Sub BtnHigher_OnClick
    document.doorbell.pitch = 880
End Sub

Sub BtnLower_OnClick
    document.doorbell.pitch = 440
End Sub

Sub BtnPlay_OnClick
    document.doorbell.buzz
End Sub
-->
</SCRIPT>
```

Now, when a user clicks a button on the Web page, the appropriate VB Script handler is invoked,

which in turn manipulates the Java applet. Note that when the applet is referenced from VBScript, its name has the identifier "document." prepended to it.

In this way, Java applets can be driven just like ActiveX controls. The script can read and write the public variables of the applet, as well as call its public methods (including passing parameters and reading return values).

Note that only the **Applet**-derived class is directly accessible from the scripting language. If your Java applet includes other classes that you want to be available to the scripting language, you must define public methods in your **Applet**-derived class that delegate to those classes.

You can also embed both Java applets and ActiveX controls in the same Web page and use VB Script handlers to connect them. For example, you can read values from an ActiveX control and pass them to a Java applet, or vice versa. (However, you cannot fire events from a Java applet.)

This automatic scripting capability is provided by the Java Support in Internet Explorer. As a result, your script code can manipulate Java applets developed with any tool, not just ones developed with Visual J++.

A more sophisticated way of having Java applets and ActiveX controls communicate is described in [Connecting a Java Applet and an OLE Control](#). This requires knowledge of [using a COM object from Java](#).

For more information about VBScript, see the VBScript Web site at <http://www.microsoft.com/vbscript/> (which is accessible using the Web Favorites command on the Help menu).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using a COM Object from Java

With compiler support for Java/COM integration, you can refer to COM objects directly from your Java source code. This makes any COM service, including those offered by programmable controls or Automation objects, available to any Java program.

- [Running the Java Type Library Wizard](#)
- [Using a COM Object in your Java Code](#)
- [Trusted vs Untrusted Applets](#)
- [What the Java Type Library Wizard Generates](#)
- [Type Mappings Between Java and COM](#)
- [COM Programming in Java and C++ Compared](#)
- [Handling COM Errors in Java](#)
- [Using the JavaTLB Command-Line Tool](#)
- [Using the OLE Object View Tool](#)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Running the Java Type Library Wizard

In order to use a COM class from Java, you must first import it; this means creating a Java class that represents the COM class in the context of Java. Visual J++ includes the Java Type Library Wizard for this purpose.

Type libraries are a mechanism defined by COM to store type information; each type library contains complete type information about one or more COM entities, including classes, interfaces, dispinterfaces, and others. For more information about type libraries, see Type Libraries and the Object Description Language, or the Win32 SDK.

{ewl msdncd, EWGraphic, com3a 0 /a "build.bmp"} To import a COM class for use in Java

- 1 From the Tools menu, choose Java Type Library Wizard.

The check-box list displays all the type libraries registered on your machine. This list reflects the entries beneath the \HKEY_CLASSES_ROOT\TypeLib key of the system registry.

- 2 Choose the type library that describes the COM class(es) you want to use.
- 3 Choose OK.

In the Output window, the Type Library Wizard prints out the **import** statement(s) appropriate for each package it has created. You can use these **import** statements in your Java source code; this allows you to refer to classes in those packages by their short names.

The Type Library Wizard also displays the complete path of a newly created file named SUMMARY.TXT. By double-clicking on this line, you can open the text file in Developer Studio. This file lists the Java signatures of all the methods for the COM interfaces and classes described by the type library. You can call these methods from your Java program.

To find out how to use the interfaces available from a particular programmable control or Automation server, consult the documentation provided by the vendor.

As an example, suppose comserver.DLL, which defines a `CComBeeper` class and an `IComBeeper` interface, is registered as having type library information. Running Java Type Library Wizard and selecting comserver.DLL would create a package named "comserver" containing the Java class `CComBeeper` and the Java interface `IComBeeper`. It would also create a SUMMARY.TXT file describing what methods that Java class and interface provide.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using a COM Object in Your Java Code

Once you've run the Java Type Library Wizard on comserver.DLL, you can use the **import** statement in your Java source code to easily refer to classes in the comserver package. For example:

```
import comserver.*;    // import comserver package
```

This is identical to Java's syntax for importing an entire package.

Note The packages and directories created by the Java Type Library Wizard always have all-lowercase names.

You could also import individual classes within a type library explicitly:

```
import comserver.IComBeeper;  
import comserver.CComBeeper;
```

Again, this is identical to Java's syntax for importing a single class.

The Java classes that wrap COM classes can be used like other Java classes. For example:

```
IComBeeper testBeep = (IComBeeper) new CComBeeper();  
int myTone = testBeep.getSound();  
testBeep.putSound(64);  
testBeep.Beep();
```

The above code declares a reference of type `IComBeeper`, initializes it by creating an instance of the COM class `CComBeeper`, gets and sets the value of a property of the object, and then calls a method on the object. For more information, see [Type Mappings Between Java and COM](#) and [COM Programming in Java and C++ Compared](#).

Important Note The only restriction on using Java classes that wrap COM classes is that you cannot use an instance of the class directly; you must always use it through an interface.

For example, the following would be legal Java code, but will cause errors when used with a COM class:

```
CComBeeper testBeep = new CComBeeper(); // DON'T DO WITH COM CLASS  
int myTone = testBeep.getSound(); // CAUSES RUN-TIME ERROR WITH COM CLASS
```

Because `CComBeeper` is a COM class, you must use one of its interfaces to manipulate an instance of the class.

COM does not support accessing properties or methods on a class independently from its interfaces.

Note that in the above example, the statement `"import comserver.*;"` is not strictly necessary. In Java, any class can be referenced using its fully qualified name (for example, `java.awt.Canvas`), as long as a class fitting that name can be found when searching the class path. The **import** statement simply allows you to refer to the class using its short name (for example, `Canvas`).

The same is true for COM classes. The **import** statement is not required in order to use the COM class, it simply allows the class to be referenced using its short name. Any COM class can always be referenced using its fully qualified name (`comserver.CComBeeper`, in the above example).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Trusted vs Untrusted Applets

Java applets typically run in a “sandbox,” a carefully delimited execution environment that prevents them from doing anything that could interfere with your system. The use of COM services means accessing resources outside the sandbox. To prevent Java applets from posing a security threat, the Java Support in Internet Explorer categorizes classes as either *trusted* or *untrusted*.

- Untrusted classes run within the sandbox and cannot use COM services. All classes that aren't loaded from the class path—which includes classes downloaded off a network—are considered untrustworthy, unless they are packaged in a .CAB (cabinet) file that has a digital signature.
- Trusted classes include any class that is loaded from the class path, and those extracted from a .CAB file that has a digital signature. Trusted classes are the only classes that are allowed to use COM services. Trusted classes are also freed of the sandbox constraints in terms of Java code; for example, you can read and write files using pure Java code, if the applet is trusted.

You can specify a .CAB file using the CABBASE parameter with the <APPLET> tag in HTML. For more information on creating a digitally signed .CAB file, see [Creating A Signed CAB File](#).

When executing a Java class, Internet Explorer behaves differently depending on whether the class is trusted or not. The primary difference is the way that Internet Explorer searches the class path.

There are four class path-related registry keys that are relevant to the security of Java applets. All of these keys are subkeys of HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM:

- Classpath - contains classes available to all applets.
- LibsDirectory - contains libraries available to all applets.
- TrustedClassPath - contains classes available only to trusted applets.
- TrustedLibsDirectory - contains libraries available only to trusted applets.

Note Everything that the Java Type Library Wizard generates is placed under the trusted library directory.

Class Path Searching During Execution

When executing a trusted applet, Internet Explorer looks in the following places for classes referenced by the applet:

1. The trusted class path.
2. The trusted library directory.
3. The class path.
4. The library directory.

When executing an untrusted applet, Internet Explorer looks in the following places for classes referenced by the applet:

1. The class path.
2. The library directory.

The Java Support for Internet Explorer also employs security mechanisms besides restricting the class path searching. For example, any classes that are Java wrappers for COM classes are inaccessible to untrusted applets, even if those classes happen to be located on the regular class path or in the libraries directory.

In summary:

- A trusted applet can access any class, no matter where it originates, whether it implements COM

services or not.

- An untrusted applet can access only pure Java classes, and only if they don't reside in the trusted class path or the trusted library directory.

Class Path Searching During Development

During compilation of any Java program, the Visual J++ compiler looks in:

1. The trusted class path.
2. The trusted library directory.
3. The class path.
4. The library directory.
5. One of the following:
 - If you're running from the command line, the class path as specified in the CLASSPATH environment variable.
 - If you're running from Developer Studio, the class path as specified in the Directories tab of the Options dialog box from the Tools menu.

Note When you launch Internet Explorer from the Developer Studio environment, the Java classes in your project directory are considered trusted. As a result, when you are developing Java applets that make use of COM services, you can easily execute them without having to package them into a signed .CAB file.

However, this does not mean that an applet built using Visual J++ automatically qualifies as trusted. Outside of Developer Studio, an applet must be packaged into a .CAB file and digitally signed in order to use COM services.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


What the Java Type Library Wizard Generates

The Java Type Library Wizard displays everything that has an entry under the registry key HKEY_CLASSES_ROOT\TypeLib. This is not restricted to .TLB files; .OLB, .OCX, .DLL and .EXE files can all contain type library information. For simplicity, all of these files are referred to as “type libraries” when discussing the Java Type Library Wizard.

For each type library imported, the Java Type Library Wizard creates a directory below the trusted library directory having the same name as the type library. The Java Type Library Wizard fills that directory with .CLASS files, one for each COM class and/or interface described in the type library. All the generated classes and interfaces are part of a Java package having the same name as the type library file. (If the type library contains an **importlib** statement, the Java Type Library Wizard creates a separate Java package in a separate directory for the imported type library.)

For instance, in the Comserver example, running the Java Type Library Wizard on comserver.DLL would create the directory \windows\java\trustlib\comserver (this assumes that \windows\java\trustlib is the trusted library directory) and place the files CComBeeper.class and IComBeeper.class there. Because the \comserver subdirectory is below the trusted library directory, the Visual J++ compiler can find the comserver subdirectory and its contents, and the statement “`import comserver.*;`” allows you to refer to the classes by their short names.

Each .CLASS file generated by the Java Type Library Wizard contains a special attribute identifying it as a wrapper for a COM class. When the Java Support in Internet Explorer sees this attribute on a class, it translates all Java method invocations on the class into COM function invocations on the COM class. (For a COM interface that defines properties, the corresponding Java interface defines two methods for each property, named **get**<property> and **put**<property>. If the COM interface defines methods with the **propget** or **propput** attributes, the Java interface versions of those methods have **get** or **put** prepended to their names. See Type Mappings Between Java and COM.)

Once you have run the Java Type Library Wizard for a given type library, you don’t need to re-run the wizard unless the type library has changed.

Once the Java Type Library Wizard has generated .CLASS files for a type library, your Java program has no further need for the type library. When distributing an applet that uses COM, your Java program needs only its own .CLASS files, the .DLL, .EXE, or .OCX that implements the COM services, and the generated .CLASS files for the COM services. You must also be sure to register the COM object appropriately.

Note From the command line, you can use the JavaTLB tool to generate Java class wrappers out of type library information. Simply type “`javatlb filename`”, where *filename* is the name of the .TLB file (or .OCX, .DLL, or .EXE file containing the type library information.) This will create the appropriate directory and .CLASS files under the trusted library directory. See Using the JavaTLB Command-Line Tool for more information.

If you include COM objects with Java applets embedded on a Web page, the .OCX or .DLL files that implement the COM object should have digital signatures. The Java applets should be packaged into a digitally signed cabinet (.CAB) file. For more information on .CAB files and digital signatures, see Creating a Signed CAB File.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Type Mappings Between Java and COM

The Java Support in Internet Explorer allows most constructs that can be specified in a type library to be accessed from Java. COM constructs that cannot be described in a type library are inaccessible from Java. To see exactly how the COM constructs are exposed in Java, see the SUMMARY.TXT files created by the Java Type Library Wizard for each class and interface.

Library Elements

There are five basic types of elements that can be defined in Object Description Language (ODL), and thus in a type library. These are mapped to Java as follows:

ODL element	Java element
coclass	public class
interface	public interface
dispinterface	public interface
typedef (struct , enum or union)	public final class
module	public final class with public static final members

The **interface** keyword in ODL is used to define custom (that is, v-table based) interfaces.

The **dispinterface** keyword in ODL is used to define dispatch interfaces. Properties in the dispatch interface are accessible in Java through two methods named **get<property>** and **put<property>** in the Java interface.

Interface Methods

The methods generated for an interface do not include those it inherits from **IUnknown** or **IDispatch**.

Methods declared with the **propget** or **propput** attributes are exposed with **get** or **put** prepended to their names.

Parameters

The following ODL types are supported, and they map to Java types in the following manner:

ODL Type	Java Type
boolean	boolean
char	char
double	double
int	int

int64	long
float	float
long	int
short	short
unsigned char	byte
BSTR	class java.lang.String
CURRENCY	long (divide by 10,000 to get the original value as a fixed-point number)
DATE	double
SCODE/ HRESULT	int (see also class com.microsoft.com.Co mException)
VARIANT	class com.microsoft.com.Va riant
IUnknown *	interface com.ms. com.IUnknown
IDispatch *	class java.lang.Object
SAFEARRAY(<i>typename</i>)	class com.microsoft.SafeArray
<i>typename</i> *	single- element array of <i>typename</i>
void	void

For information on accessing the contents of a VARIANT structure, see the class [Variant](#).

IUnknown is the interface from which all COM interfaces are derived. The Java versions of COM interfaces are derived from **com.ms.com.IUnknown**. You never have to call any methods on

com.ms.com.IUnknown. If your Java program calls a COM method that takes a parameter of type **com.ms.com.IUnknown**, you can pass any COM interface. If your Java program calls a COM method that has **com.ms.com.IUnknown** as its return type, you can cast the return value to the COM interface you're expecting.

Certain ODL attributes cause a parameter to be treated in a special manner:

- VARIANT parameters marked with the **optional** attribute in ODL are exposed as ordinary parameters in Java. If you wish to omit a parameter when calling the method, call the noParam method on a Variant object and pass that as a "placeholder."
- A parameter marked with the **retval** attribute in ODL is treated as the return value in the corresponding Java method.

The following are examples of ODL declarations and their corresponding Java declarations.

Declarations with simple parameters:

```
HRESULT void
T      Func(int
Func([in] x);
int x);

HRESULT void
T      Func(int[]
Func([in, x);
out] int*
x);

HRESULT int Func();
T
Func([out
,retval]
int* x);
```

Declarations with string parameters:

```
HRESULT void
T      Func(java
Func([in] .lang.String
BSTR x); g x);

HRESULT void
T      Func(java
Func([in, .lang.String
out] g[] x);
BSTR*
x);

HRESULT java.lang.
T      String
Func([out Func();
,retval]
BSTR*
x);
```

Declarations with VARIANT parameters:

```
HRESULT void
```



```

Func([in]      Func(co
VARIANT      m.ms.c
x);           om.Vari
              ant x);

HRESULT void
Func([in]      Func(co
VARIANT*      m.ms.c
x);           om.Vari
              ant x);

HRESULT void
Func([in,ou    Func(co
t]            m.ms.c
VARIANT*      om.Vari
x);           ant x);

HRESULT com.ms
Func([out,r    .com.Va
etval]        riant
VARIANT*      Func();
x);

HRESULT void
Func([in,ou    Func(co
t]            m.ms.c
VARIANT**     om.Vari
x);           ant[] x);

```

Declarations with interface pointer parameters:

```

HRESULT void
Func([in]      Func(IB
IBar* x);      ar x);

HRESULT void
Func([in,o     Func(IB
ut] IBar**     ar[] x);
x);

HRESULT IBar
Func([out,     Func();
retval]
IBar** x);

```

For information on the HRESULT returned by a COM function, see [Handling COM Errors in Java](#).

For more information about type libraries, see [Type Libraries and the Object Description Language](#), or the OLE SDK.

You can also use the OLE Object View tool to browse a type library and learn more about what was in the original ODL declaration; see [Using the OLE Object View Tool](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


COM Programming in Java and C++ Compared

For those who are familiar with COM programming in C++, this topic compares it with COM programming in Java. Java's notion of multiple interfaces on an object, along with Java's garbage collection, map readily to COM's paradigm for managing objects. For more information on COM, see the Component Object Model Specification in the Win32 Software Development Kit.

Object Allocation

In C++, you allocate a new COM object using syntax like the following:

```
IDrawable pDrawable;  
CoCreateInstance(CLSID_MyCircle, NULL, CLSCTX_SERVER,  
                IID_IDrawable, (void**) &pDrawable );
```

In Java, the equivalent code would look like this:

```
IDrawable drawable = (IDrawable) new MyCircle();
```

This is identical to Java's syntax for allocating ordinary Java objects. Behind the scenes, this line performs a call to the COM API function **CoCreateInstance** instead of simply allocating space in the run-time heap. However, that is invisible to the Java programmer.

Important Note Note that when using Java classes that wrap COM classes, you cannot use an instance of the class directly; you must always use it through an interface. For example, the following code should not be used:

```
MyCircle circ = new MyCircle(); // DON'T DO WITH COM CLASS
```

You can use `MyCircle` only through one of the interfaces that it supports. Otherwise, you will get a run-time error when you try and use the class itself.

Changing Interfaces

COM objects must implement the **IUnknown::QueryInterface** function to allow clients to switch between the object's supported interfaces. In Java, the details are handled by the Java Support for Internet Explorer, so at the source-code level it is simply done with a typecast.

For example, in C++ you might have code like this:

```
// pDrawable points to an object that supports  
// both IDrawable and IPrintable  
IPrintable *pPrintable;  
pDrawable->QueryInterface( IID_IPrintable,  
                          (void **) &pPrintable );
```

In Java, the equivalent code would look like this:

```
// drawable refers to an object that supports  
// both IDrawable and IPrintable  
IPrintable printable;  
printable = (IPrintable) drawable;
```


If the object does not implement the requested interface, a **ClassCastException** is thrown when you attempt the cast. You can also use the **instanceof** operator to check beforehand whether the object implements the requested interface. Again, this is identical to Java's syntax for casting between ordinary Java object types.

Object Identity

In COM, object identity is defined as having the same **IUnknown** pointer. For example, in C++, to check whether two pointers refer to the same object, you need code like this:

```
// comparing pDrawable and pPrintable
IUnknown *pUnk1, *pUnk2;
pDrawable->QueryInterface( IID_IUnknown,
                           (void **)&pUnk1 );
pPrintable->QueryInterface( IID_IUnknown,
                           (void **)&pUnk2 );
if ( pUnk1 == pUnk2 )
    printf( "It's the same object" );
```

In Java, the equivalent code would look like this:

```
// comparing drawable and printable
if (drawable == printable )
    System.out.println( "It's the same object" );
```

As in the previous examples, this is identical to Java's syntax for comparing two object references.

Reference Counting

Reference counting is handled automatically in Java. There is no need to call **IUnknown::AddRef** when creating a new reference to an object, nor do you need to call **IUnknown::Release** when you're finished using a reference to an object. The Java garbage collector automatically keeps track of how many references there are to an object.

Reuse

In C++, you can extend the functionality of an existing COM class through aggregation. In Java, you can do the same simply using the **extends** clause, just as you would extend an ordinary Java class. However, because Java uses a single inheritance model, you can extend only a single COM class, even though COM allows multiple aggregation. (In the rare situation where the original COM class does not support aggregation, extending the class will not work; a **ComException** object is thrown when you try allocating an instance of the derived class. See [Handling COM Errors in Java](#) for more information.)

Summary

As shown by these examples, COM programming is much simpler in Java than in C++. The Java Support in Internet Explorer performs all the tedious work for you, making COM classes as easy to use as native Java classes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Handling COM Errors in Java

COM methods typically return a value known as an HRESULT, which is a 32-bit error code. The Java Support for Internet Explorer defines a class called [com.ms.com.ComException](#). This class wraps the HRESULT error code, and is used to communicate error information from COM back to Java whenever a COM method fails.

Whenever a COM method is exposed to Java, it is exposed as a Java method with an implicit **throws** clause. For example:

```
int convert( char x ) throws com.ms.com.ComException;
```

Because **ComException** is derived from the Java class **RuntimeException**, the compiler does not strictly require a **throws** clause in the method declaration. Nor does the compiler require **try** and **catch** blocks; run-time exceptions are unchecked by the compiler, so it is up to your discretion when to use **try-catch** blocks.

The **ComException** class defines a [getHRESULT](#) method that returns the error code, in the form of a Java **int**, describing the specific error. You can also use the **getMessage** method (defined by the Java class **Throwable**) to get the detail message.

There are two subclasses of **ComException**: **ComFailException** and **ComSuccessException**. **ComFailException** is the one you typically try to catch.

```
try
{
    // call COM methods
}
catch (com.ms.com.ComFailException e)
{
    System.out.println( "COM Exception:" );
    System.out.println( e.getHRESULT() );
    System.out.println( e.getMessage() );
}
```

Note that the **ComFailException** class is referenced using its fully-qualified name. If you added an `"import com.ms.com.*;"` statement to the beginning of the file, you could refer to the class by its short name.

For a list of the values of common system-defined HRESULTs, see [ComFailException](#). For domain-specific errors, consult the documentation for the component you are using.

The documentation for many components describes errors in terms of Visual Basic **Err.Number** values rather than HRESULTs. However, some components use a convention where there is a correspondence between the HRESULT returned and the **Err.Number** value. For example, if you are using Data Access Objects (DAO) or Remote Data Objects (RDO), you can call the **getHRESULT** method on the **ComFailException** object, take the lower 16 bits of the returned HRESULT, convert that value to decimal, and check it against the errors described in the DAO or RDO documentation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the JavaTLB Command-Line Tool

The JavaTLB tool generates .CLASS files for the classes and interfaces described by a type library. These .CLASS files allow COM services to be used by Java programs. The JavaTLB tool can be used from within Developer Studio by running the Java Type Library Wizard.

The typical usage is as follows:

```
javatlb filename
```

where *filename* is the name of the type library (.TLB, .OLB, .OCX, .DLL or .EXE).

For example:

```
javatlb widgets.tlb
```

This command first creates a \widgets subdirectory underneath the trusted library directory specified by the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM\TrustedLibsDirectory. If the trusted library directory is \windows\java\trustlib, the above command creates the directory \windows\java\trustlib\widgets, and fills it with .CLASS files, one for each class or interface described by the widgets.TLB type library.

If the type library contains an **importlib** statement, JavaTLB creates a separate Java package in a separate directory for the imported type library.

Note The packages and directories created by JavaTLB always have all-lowercase names.

You can also use a response file to run JavaTLB on multiple type libraries at once. For example:

```
javatlb @list
```

The filename after the @ must specify a text file containing the name of the type library files.

Options

JavaTLB also supports certain command-line options:

/U *classname*

This option causes JavaTLB to display the signatures for all the public methods in the .CLASS file. For example:

```
javatlb /U IDooHickey.class
```

This command would produce output similar to the following:

```
public interface widgets/IDooHickey extends java.lang.Object
{
    public native short getThingie();
    public native void putThingie(short);
    public native void doWhatever();
}
```

This describes the methods you can call when using the IDooHickey interface. There are several

things worth noting about the output:

- In the declaration of the class or interface, the name *packagename/classname* format is used strictly to provide information about what package the class or interface belongs to. An identifier like `widgets/IDooHickey` is not legal in Java.
- In general, all methods can throw an instance of `com.ms.com.ComException`.
- Only interfaces have methods; classes do not. When using the Java wrappers for COM classes, you must always call methods through an interface, not through the class itself.

For information on how COM methods and parameters are translated into Java, see Type Mappings Between Java and COM.

/U:T

This option generates .CLASS files and then writes the signature information to a text file. For example, running the command:

```
javatlb /U:T widgets.tlb
```

is the equivalent of first running the command “`javatlb widgets.tlb`” and then running the command “`javatlb /U *.class > summary.txt`” in the `/widgets` subdirectory.

This option is what is specified by the Java Type Library Wizard.

/p package

This option lets you specify where JavaTLB will place the .CLASS files it generates. For example:

```
javatlb /p gremlinsoft widgets.tlb
```

This places the .CLASS files in the directory `\windows\java\trustlib\gremlinsoft\widgets`. This option allows you to define, for example, a company-specific directory tree that contains the .CLASS files for all the COM objects produced by your company.

The *package* parameter can have multiple levels; for example, the command “`javatlb /p gremlinsoft.toolworks widgets.tlb`” would place the .CLASS files in the directory `\windows\java\trustlib\gremlinsoft\toolworks\widgets`. To use these classes in your Java program, you would need to specify an **import** statement of the form “`import gremlinsoft.toolworks.widgets.*;`” in your source code.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the OLE Object View Tool

To find out more information about the methods exposed by a particular programmable control or Automation server (beyond what is described in the output of the Java Type Library Wizard or the JavaTLB tool), consult the documentation provided by the vendor.

If you don't have complete documentation, you can also use the OLE Object View tool. This tool can provide information about what attributes a particular method has, as well as what parameter names were specified in the original ODL definition.

{ewl msdncd, EWGraphic, com23a 0 /a "build.bmp"} To use the OLE 2 Object Viewer

- 1** From the Tools menu, choose OLE Object View.
The OLE 2 Object Viewer appears. The pane on the left side is the Object List pane.
- 2** Double-click on the type library you want to view.
The ITypeLib Interface Viewer appears.
- 3** From the TypeInfo drop-down list, choose the class or interface you want information on.
Classes have the entry "typekind = coclass" in the summary window on the right side. Interfaces have the entry "typekind = interface" or "typekind = dispinterface" in the summary window.
When an interface is selected in the drop-down list, the Functions box lists all the methods defined by that interface. The Variables/Data Members box also lists any properties defined by the interface.
- 4** Highlight a method in the Functions list box.
The Function Prototype box displays the return type and parameter list for the highlighted method.
The FUNCDESC/VARDESC box display attributes for the highlighted method.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Connecting a Java Applet and an OLE Control

You can have a Java applet interact with an OLE control that resides on the same HTML page, using a combination of the techniques described in the topics Controlling a Java Applet through Scripting and Using a COM object from Java.

There are two distinct ways of connecting a Java applet with an OLE control:

- Forwarding Events to a Java Applet
This allows the applet to respond to events fired by the control.
- Having the applet invoke methods or set properties on the control
This is done by passing the control to the Java applet. This requires work in both Java and VBScript:
 - The receiving end (Java)
 - The sending end (VBScript)

It is common to use both types of connection in order to establish two-way communication between the applet and the control.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Forwarding Events to a Java Applet

A Java applet cannot directly receive events fired by an OLE control. However, you can define VBScript methods that handle events fired by the control. These methods can then, in turn, call methods on the Java applet, as described in [Controlling a Java Applet through Scripting](#).

For example, consider an HTML page containing both an OLE control and a Java applet:

```
<OBJECT ID=Outline WIDTH=100 HEIGHT=100
  CLASSID="CLSID:BE4F3AC5-AEC9-101A-947B-00DD010F7B46">
</OBJECT>
```

...

```
<APPLET CODE="OutlineUser.class" ID=user>
```

This page contains an outline control and an applet that uses the outline control.

You can then define event handlers in VBScript for the events fired by the control. For example:

```
<SCRIPT language=VBScript>
<!--
sub Outline_Collapse
  document.user.handleCollapse
end sub

sub Outline_Expand
  document.user.handleExpand
end sub
-->
</SCRIPT>
```

In this way, VBScript acts as an intermediary between the OLE control and the Java applet, receiving events from the former and calling methods on the latter.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Receiving a Control as a Parameter in Java

You cannot embed an OLE control directly into a Java applet the way you embed an AWT (Abstract Window Toolkit) control. However, you can have your HTML page pass an OLE control as a parameter to a Java method, and have your Java applet use it that way.

In order for your Java applet to know how to use an OLE control, you must first use the Java Type Library Wizard to import the OLE control, that is, to generate .CLASS files that expose the control's functionality to Java programs.

In your Java source, you should use one or more **import** statements to allow convenient references to the package associated with the OLE control. For example, if you had run the Java Type Library Wizard on the MSOUTL32.OCX control, you would use statements like this:

```
import msoutl32.*;
import olepro32.*;
```

There are two **import** statements because the type library inside MSOUTL32.OCX contains an **importlib** statement.

Then in your Java applet, define a public method that takes an **Object** as a parameter. For example:

```
public OutlineUser extends Applet
{
    // IOutlineCtrl interface defined in package MSOUTL32
    IOutlineCtrl m_outlineCtrl;

    public void setCtrl(Object oc)
    {
        m_outlineCtrl = (IOutlineCtrl)oc;
    }

    // other methods
    // ...
}
```

The `setCtrl` method casts the object to the `IOutlineCtrl` interface and stores a reference to that object. Because this method is public, it's accessible to the VBScript portion of the HTML page in which the applet resides. A VBScript function will use this function to pass an OLE control to the applet.

Once your Java applet has a reference to the object, in the form of an appropriate interface, you can call methods on the outline control. For example:

```
public String getText()
{
    return m_outlineCtrl.getText();
}
```

This Java method retrieves a property from the outline control.

Note that only trusted Java applets can make use of OLE controls in this manner.

Preparing the Java applet to receive the control is only half the work needed to let the applet drive an OLE control. The other half is passing the control from VBScript.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Passing a Control from VBScript

First insert the OLE control on your HTML page using a tag like the following:

```
<OBJECT ID=Outline WIDTH=100 HEIGHT=100  
  CLASSID="CLSID:0713E8A2-850A-101B-AFC0-4210102A8DA7">  
</OBJECT>
```

This tag embeds an instance of the a outline control (which is implemented by MSOUTL32.OCX). The NAME attribute lets you refer to the button from the scripting language. You can use the ActiveX Control Pad to insert controls without having to enter the CLSID manually.

Then insert the Java applet on the HTML page using a tag like the following:

```
<APPLET CODE="OutlineUser.class" ID=user>
```

The ID attribute lets you refer to the applet from the scripting language.

In the VBScript portion of your HTML page, define a **window_onLoad** function that passes a reference to the OLE control to the Java applet. For example:

```
<SCRIPT language="VBScript">  
<!--  
sub window_onLoad  
  document.user.SetCtrl Outline  
end sub  
-->  
</SCRIPT>
```

The **window_onLoad** function is called as soon as the HTML page has finished loading. The above example causes a reference to the outline control to be passed to the Java applet's `setCtrl` method, as defined in [Receiving a Control as a Parameter in Java](#). From that point forward, the Java applet can manipulate the outline control.

By both passing the control to the Java applet, and forwarding events from the control to the applet, you establish two-way communication between the control and the applet.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Exposing a Java Class as a COM Class

Besides allowing Java programs to use COM objects, the Java Support in Internet Explorer also allows Java programs to expose their functionality as COM services. This lets you use Java for developing component software without requiring that your clients use Java; your clients can use any language that is compatible with COM, such as Microsoft Visual Basic or C++.

- [Describing the Interfaces on the COM Class](#)
- [Writing the Java Class](#)
- [Registering the Java Class as a COM Class](#)
- [Returning HRESULTs from Java](#)
- [Using JavaReg](#)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Describing the Interfaces on the COM Class

The first step is to use Object Description Language (ODL) to define how your Java class will be exposed. For example, you might define a COM class that implements two interfaces. Define the interfaces in an ODL file:

```
// mytroupe.odl
[ uuid (42E5AFC3-DBAA-11cf-BAFD-00AA0057B223) ]
library LMyTroupe
{
    importlib("stdole32.tlb");

    [ odl, uuid(42E5AFC4-DBAA-11cf-BAFD-00AA0057B223) ]
    interface ICanSing
    {
        HRESULT sing();
        HRESULT hum();
    }

    [ odl, uuid(42E5AFC5-DBAA-11cf-BAFD-00AA0057B223) ]
    interface ICanDance
    {
        HRESULT dance();
        HRESULT shuffle();
    }
    [ uuid(42E5AFC6-DBAA-11cf-BAFD-00AA0057B223) ]
    coclass MyPerformer
    {
        interface ICanSing;
        interface ICanDance;
    }
};
```

You can expose your Java interfaces through COM as interfaces, dispatch interfaces, or dual interfaces; dual interfaces are recommended. Services that cannot be described in a type library cannot be exposed by your Java program.

Use MkTypLib or the MIDL compiler to build mytroupe.ODL into mytroupe.TLB. (If you don't have Visual C++ installed, you must specify the **/nocpp** option on the command line when running these tools, and make sure you do not use any preprocessor directives in your .ODL file.) For more information about writing ODL files and using MkTypLib or the MIDL compiler, see [Type Libraries and the Object Description Language](#), or the OLE SDK.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Writing the Java Class

Then use the JavaTLB tool to generate .CLASS files based on the type library.

Next, create a .JAVA source file containing the Java implementation of the class. Declare a Java class that implements all the interfaces supported by the COM class in the .ODL file. The Java class is not required to have the same name that was used in the **coclass** statement, but using the same name will make your code more readable. For example:

```
import com.ms.com.*;
import mytroupe.*;

class MyPerformer implements ICanSing, ICanDance
{
    public void sing() throws ComException {}
    public void hum() throws ComException {}
    public void dance() throws ComException {}
    public void shuffle() throws ComException {}
}
```

The Visual J++ compiler will generate an error if the class does not implement all the methods defined by the interfaces specified in the .ODL file. (Note that the “import com.ms.com.*;” statement lets you refer to ComException by its short name.)

Compile the implementation .JAVA file into a .CLASS file. COM clients can read the .TLB file to determine what services your class exposes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Registering the Java Class as a COM Class

Finally, run the JavaReg tool from the command line to register your Java class as a COM class. Specify the name of the Java class and the CLSID that you want to register it under. For example:

```
javareg /register /class:MyPerformer /clsid:{42E5AFC6-DBAA-11cf-BAFD-00AA0057B223}
```

The CLSID you specify on the command line must match the one specified for the COM class in the .ODL file. It is this CLSID which associates the COM class defined in the .ODL file with the Java class you wrote.

If the Java class you wrote is part of a Java package (that is, if you used the **package** keyword in the Java source code), you must specify the package name along with the class name when using the **/class** option.

The JavaReg tool creates registry entries that distinguish a COM class written in Java from other COM classes. The most important entries are the following:

```
HKEY_CLASSES_ROOT
  CLSID
    {42E5AFC6-DBAA-11cf-BAFD-00AA0057B223}
      InprocServer32 = msjava.dll
      JavaClass = MyPerformer
```

When an object of that CLSID is requested, the Java Support in Internet Explorer is launched and the specified class is loaded. The run time exports a **DllGetClassObject** function and creates a COM class factory for the Java class. This means that a COM client doesn't need to know that a given class is implemented in Java. To create an instance of the class, the client simply passes the CLSID to **CoCreateInstance**, just as it would with any other COM class.

Note that for all COM classes implemented in Java, the value of the **InprocServer32** key is identical: MSJAVA.DLL, which contains the Java Support for Internet Explorer. What differs for each class is the value named **JavaClass** beneath the **InprocServer32** key.

If you distribute a COM class written in Java, you must provide an installation program to add the appropriate entries to the registry (or invoke JavaReg to do so). In addition, your installation program should install the .CLASS file on the class path of the client machine so that the Java Support for Internet Explorer can find it.

Note that while any COM class qualifies as an ActiveX control, a COM object must implement many more interfaces in order to be a full-fledged control that can appear in tool palettes and be inserted in control containers.

Java classes exposed as COM classes are automatically aggregatable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Returning HRESULTs from Java

Throw an instance of [com.ms.com.ComFailException](#) to indicate failure. You can specify a particular HRESULT when constructing the **ComFailException** object. The HRESULT will be used as the return value for the COM method. (For a list of the values of common system-defined HRESULTs, see [ComFailException](#). For a complete list of system-defined HRESULT values, see the header file WINERROR.H included with the Win32 SDK.)

To indicate successful completion, you don't need to do anything; just return normally. To return S_FALSE (indicating a successful completion but a return value of Boolean FALSE), throw an instance of [com.ms.com.ComSuccessException](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using JavaReg

Many of JavaReg's options are intended for use only with Distributed COM (DCOM), available with Windows NT version 4.0. For more information about DCOM see the Win32 Software Development Kit.

Remoting with DCOM

The Java Support in Internet Explorer runs as an in-process server, and in-process servers cannot normally be remoted using the Windows NT 4.0 Distributed COM (DCOM). However, it is possible to launch a "surrogate" .EXE in its own process that then loads the in-process server. This surrogate can then be remoted using DCOM, in effect allowing the in-process server to be remoted.

You can use JavaReg's **/surrogate** option to support remote access to a COM class implemented in Java. When first registering the class, specify the **/surrogate** option on the command line. For example:

```
javareg /register /class:MyPerformer /clsid:{42E5AFC6-DBAA-11cf-BAFD-00AA0057B223} /surrogate
```

This adds a **LocalServer32** key to the registry in addition to the usual **InprocServer32** key. The command line under the **LocalServer32** key specifies JavaReg with the **/surrogate** but without the **/register** option.

```
HKEY_CLASSES_ROOT
  CLSID
    {42E5AFC6-DBAA-11cf-BAFD-00AA0057B223}
      InprocServer32 = msjava.dll
      LocalServer32 = javareg /clsid:{42E5AFC6-DBAA-11cf-BAFD-00AA0057B223} /surrogate
```

This causes JavaReg to act as the surrogate itself. When a remote client requests services from the COM class that you've implemented using Java, JavaReg is invoked. JavaReg then loads the Java Support in Internet Explorer with the specified Java class. (This means that when distributing your Java program, your installation program must install JavaReg along with the Java class.)

You can remove the **LocalServer32** key by rerunning JavaReg with the **/class** option, specifying the same class name, but without the **/clsid** or **/surrogate** options.

Using JavaReg to Generate a CLSID

If you aren't defining new interfaces, but are simply writing a Java class that implements existing interfaces, you may not need to write an .ODL file. First run the Java Type Library Wizard over the type library that describes the COM interfaces you want to implement; this generates .CLASS files for those interfaces. Then write a Java class that implements those COM interfaces.

Finally, run JavaReg with the **/register** and **/class** options, but without the **/clsid** option. This instructs JavaReg to generate a new CLSID and register the class using that value. You must record the CLSID that JavaReg generates, since this is the value you must publish when distributing your class. As long as the interfaces your class implements are properly registered, COM clients can specify your class's CLSID and use its interfaces as usual.

Other JavaReg Options

JavaReg also accepts other command-line options:

/unregister

Use this option to remove the registry entries for a particular class. You must include the **/class** option to specify the name of the class.

/progid

Use this option if you want to specify a ProgID for your class.

/defappid

This option is only meaningful if you are using the **/surrogate** option. By default, JavaReg uses the CLSID as the AppID. By specifying this option, JavaReg will use its own CLSID as the AppID. For more information on the AppID, see the Win32 Software Development Kit.

/single

This option is only meaningful if you are using the **/surrogate** option. This option causes a single instance of your class to be shared among multiple clients. Use this option if your object has no instance data, or if you want its instance data to be shared by all clients.

/console

This option is only meaningful if you are using the **/surrogate** option. This option creates a console window that displays the **System.out** output from your Java applet. This is useful for debugging purposes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating a Signed CAB File

The cab&sign directory on the Visual J++ CD-ROM contains two tools:

- CabDevKit.EXE
- CodeSignKit.EXE

Both of these are self-extracting executables that install the tools and documentation needed to, respectively, create a Cabinet (.CAB) file, and digitally sign a file.

These executables are not installed by the Visual J++ setup program. You need to copy each of these executables into its own directory and run them to extract their respective tool kits.

You can also get all the tools needed for creating a .CAB file and digitally signing a file as part of the ActiveX Software Development Kit. You can download the ActiveX SDK from <http://www.microsoft.com/activex/> (which is accessible using the Web Favorites command on the Help menu).

See the respective documentation included with the tool kits for complete information on using these tools.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Updating the Java Support on a User's Machine

If you are placing an applet that uses COM on an HTML page accessible from the Internet, you must ensure that any users who encounter that page have a version of the Java Support for Internet Explorer that fully supports Java/COM integration.

To do this, you must insert the following tag on the HTML page containing your applet (or on the introductory page of your Web site):

```
<OBJECT  
CLASSID="clsid:08B0E5C0-4FCB-11CF-AAA5-00401C608500"  
CODEBASE="http://www.microsoft.com/java/IE30Java.cab#Version=1,0,0,1">  
</OBJECT>
```

This tag causes the user's Internet Explorer to check the version of its Java support. If the version installed on the user's machine is not up-to-date, Internet Explorer downloads the latest version of Java support from <http://www.microsoft.com> and updates the user's machine.

An example of this tag can be found in the \MSDEV\REDIST directory of your Visual J++ installation, in the file ObjectTag.HTML.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Type Libraries and the Object Description Language

When you expose OLE Automation objects, it allows interoperability with the programs of other vendors. For vendors to use these objects, they must have access to the characteristics of the objects (properties and methods). To make this information available:

- Publish object and type definitions (for example, as printed documentation).
- Code objects into a compiled .c or .cpp file so they can be accessed using **IDispatch::GetTypeInfo** or implementations of the **TypeInfo** and **TypeLib** interfaces.
- Use the MIDL compiler or the MkTypLib utility to create a type library that contains the objects, and then make the type library available.

The Microsoft Interface Definition Language (MIDL) compiler and the MkTypLib utility both compile scripts that are written in the Object Description Language (ODL). Microsoft has expanded the Interface Definition Language (IDL) to contain the complete ODL syntax.

For more information about the MIDL compiler, refer to the *MIDL Programmer's Guide and Reference* in the Win32 SDK.

The following descriptions and references are contained in this chapter:

- Contents of a type library
- Using MIDL and MkTypeLib
- MkTypLib type library creation
- ODL file syntax
- ODL reference

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Contents of a Type Library

Type libraries are OLE compound document files (.tlb files) that include information about types and objects exposed by an OLE application. A type library can contain any of the following:

- Information about data types, such as aliases, enumerations, structures, or unions.
- Descriptions of one or more objects, such as a module, interface, **IDispatch** interface (dispinterface), or component object class (coclass). Each of these descriptions is commonly referred to as a *typeinfo*.
- References to type descriptions from other type libraries.

By including the type library with a product, the information about the objects in the library can be made available to the users of the applications and programming tools. Type libraries can be shipped in any of the following forms:

- A resource in a dynamic link library (DLL). This resource should have the type TypeLib and an integer ID. It must be declared in the resource (.rc) file as follows:

```
1 typelib mylib1.tlb
2 typelib mylib2.tlb
```

There can be multiple type library resources in a DLL. Application developers should use the resource compiler to add the .tlb file to their own DLL. A DLL with one or more type library resources typically has the file extension .olb (object library).

- A resource in an .exe file. The file can contain multiple type libraries.
- A stand-alone binary file. The .tlb (type library) file output by the MkTypLib utility is a binary file.

Object browsers, compilers, and similar tools access type libraries through the interfaces **ITypeLib**, **ITypeInfo**, and **ITypeComp**. Type library tools (such as MkTypLib) can be created using the interfaces **ICreateTypeLib** and **ICreateTypeInfo**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using MIDL and MkTypeLib

Files parsed by MkTypeLib are .odl files. Files parsed by MIDL are referred to as .idl files, although they can contain the same syntax elements as .odl files. The MIDL compiler and the MkTypeLib utility both compile scripts written in the Object Description Language (ODL).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding ODL to an IDL Definition

The .odl files provide definitions that are added to a type library. MkTypeLib parses files written in the ODL syntax, generates type libraries, and optionally creates C++ header files that contain the same definitions.

The top-level element of the ODL syntax is the **library** statement (or library block). Every other ODL statement (with the exception of the attributes that can be applied to the **library** statement) must be defined in the library block.

The MIDL function generates a type library when it sees a **library** statement in the same way that MkTypeLib does. The statements found in the library block follow essentially the same syntax as earlier versions of the ODL language.

ODL attributes can be applied to an element both inside and outside of the library block. Outside the block, they typically do nothing, unless the element is referenced from within the block by using it as a base type, inheriting from it, or referencing it on a line such as this:

```
// IDL definitions including definitions for interface
// and struct bar.
// Attributes omitted here for brevity.

library a
{
    interface;
    struct bar;
    ...
};
```

If an element defined outside of the block is referenced in the block, its definition is put into the generated type library.

Anything outside of the library block is an .idl file, and the MIDL compiler processes it as usual. Typically, this means generating remoting stubs for it.

Support for ODL Base Types

There are a number of base types supported by MkTypeLib that are not directly supported by MIDL. The MIDL function gets its definitions for these base types by automatically importing Oleauto.idl, and Oleidl.idl whenever it encounters a library statement. This means that Oleauto.idl and Oleidl.idl (along with the imported Unknwn.idl and Wtypes.idl files) must be somewhere in the user's INCLUDE path. The OLE and OLE Automation DLLs must also be in the system if the user compiles an .idl file that contains a library statement.

The following table contains the command line options for the MIDL compiler:

Option	Description
--------	-------------

/tlb	Name of the <file type library file name> output .tlb file. If not specified, this is the same as the name of the .odl file
------	---

with the extension of .tlb.

/h Similar to the **<file /header** in nam MIDL, but also containing definitions related to the type library.

/ **<system>** is **<sy win16, win32, ste mac, mips, m> alpha, ppc, or ppc32.**

/ Same as the **align /Zp** switch in n MIDL. It sets **<#>** the alignment for types in the library.

/o Redirects **<out output. put file>**

/ Disables the **no! display of the ogo** copyright logo.

/ Same as the **noc /nocpp** switch **pp** in MIDL.

/ Puts MIDL into **mkt MkTypLib- ypli compatibility b20** mode.

3

/mktypelib203 Option

The MIDL compiler behaves differently from the MkTypLib utility. The **/mktypelib203** option removes most of these differences and makes MIDL act like MkTypLib version 2.03.

For example, **BOOL** (a MkTypLib base type) is defined differently in MIDL than it is in MkTypLib. MkTypLib treats **BOOL** as a **VARIANT_BOOL**. However, **BOOL** is defined in the file **Wtypes.idl** as a long data type. If a **VARIANT_BOOL** is to be placed in the type library, it has to explicitly use **VARIANT_BOOL** in the IDL/ODL. If **BOOL** is used when **VARIANT_BOOL** is meant to be used, then the **/mktypelib203** option should also be used.

MIDL normally puts GUID predefinitions in its generated header files, and only puts GUID instantiations in the file generated by the **/iid** option. With the **/mktypelib203** option, MIDL defines GUIDs in the header files in the way that MkTypLib does. They are defined with a macro that can be compiled conditionally to generate either a predefined or an instantiated GUID.

With the **/mktypelib203** option enabled, it is invalid to put any statements outside of the library block. A pure ODL syntax must be used; it cannot be mixed and matched in this mode.

MkTypLib is used to require **structure**, **union**, and **enum** to be defined as part of type definitions.

For example:

```
typedef struct foo { int i; } bar;
```

In this statement, MkTypLib generates a TKIND_RECORD named "bar." Because the "foo" was not recorded anywhere in the type library, it can be omitted.

MIDL allows normal C definitions of **structure**, **union**, and **enum**:

```
struct foo {int i;};  
    typedef struct foo bar;
```

- Or -

```
    typedef struct foo {int i;} bar;
```

This statement generates a TKIND_RECORD named "foo" and (if the type definition is public) a TKIND_ALIAS named "bar." The "foo" can still be omitted, in which case MIDL generates a name for it.

When the **/mktypelib203** option is enabled, the original MkTypLib type definition syntax is required for structures, unions, and enumerators. The behavior is the same as under MkTypLib (that is, "foo" is not included in the type library).

Note MkTypLib permits some scoping errors, such as giving enumerators their own scope. These errors are fixed by MIDL, and cannot be reintroduced, even with the **/mktypelib203** switch. Even though the **/mktypelib203** switch enables MIDL to compile most earlier .odl files, there can be a few exceptions. These are cases where the .odl files were already broken, and MkTypLib did not catch the errors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MkTypLib: Type Library Creation Tool

MkTypLib processes scripts written in the Object Description Language (ODL), producing a type library and an optional C or C++ header file.

MkTypLib uses the **ICreateTypeLib** and **ICreateTypeInfo** interfaces to create type libraries. Type libraries can then be accessed by tools, such as type browsers and compilers that use the **ITypelib** and **TypeInfo** interfaces, as shown in the following figure.

```
{ewc msdncd, EWGraphic, com5b 0 /a "com_01.BMP"}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Invoking MkTypLib

To invoke MkTypLib, click the Windows® 95 **Start** button, and then click **Run**. Enter the following command line in the **Open** box:

MkTypLib [*options*] *ODLfile*

MkTypLib creates a type library (.tlb) file based on the object description script in the file specified by *ODLfile*. It can optionally produce a header (.h) file, which is a stripped version of the input file. This file is included in C or C++ programs that want to access the types defined in the input file. In the header file, MkTypLib inserts DEFINE_GUID macros for each element defined in the type library (such as interface, dispinterface, and so on).

There can be a series of options, each prefixed with a hyphen (-) or a slash (/), as follows:

Option Description	
<hr/>	
/?	Displays command line Help. In this case, <i>ODLfile</i> does not need to be specified.
/align: <i>alignment</i>	Sets the default alignment for types in the library. An <i>alignment</i> value of 1 indicates natural alignment; <i>n</i> indicates alignment on byte <i>n</i> .
/cpp_ <i>cmd</i> <i>cpppath</i>	Specifies the command to run the C preprocessor. By default, MkTypLib invokes CL.
/cpp_ <i>opt</i> " <i>options</i> "	Specifies options for the C preprocessor. The default is /C /E /D__MkTypLib__.
/D	Defines the

defin name *define*
e[=val for the C
ue] preprocessor.
The *value* is
its optional
value. No
space is
allowed
between the
equal sign (=)
and the
value.

/h Specifies
filena *filename* as
me the name for
a stripped
version of the
input file. This
file can be
used as a C
or C++
header file.

/I Specifies
includ *includedir* as
edir the directory
where
include files
are located
for the C
preprocessor.

/ Suppresses
nocp invocation of
p the C
preprocessor.
Specify this
option if you
do not have
the C
compiler
installed on
your
machine.
(This means
that you
cannot use
preprocessor
directives in
your .ODL
file.)

/ Disables the
nolog display of the
o copyright
banner.

/o Redirects
outpu output (for

tfile example,
error
messages) to
the specified
outputfile.

/tlb Specifies
filename as
me the name of
the output .tlb
file. If not
specified, it
will be the
same name
as the
ODLfile, with
the extension
.tlb.

/win16
/win32
/mac
/mips
/alpha

/ Specifies the
ppc / output type
ppc3 library to be
2 produced.
The default is
the current
operating
system.

/w0 Disables
warnings.

Although MkTypLib offers minimal error reporting, error messages include accurate line number and column number information that can be used with text editors to locate the source of errors.

MkTypLib spawns the C preprocessor. The symbol **__MKTYPLIB__** is predefined for the preprocessor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ODL File Syntax

The general syntax for an .odl file is as follows:

```
[attributes] library libname {definitions};
```

The *attributes* associate characteristics with the library, such as its Help file and UUID. Attributes must be enclosed in square brackets.

The *definitions* consist of the descriptions of the imported libraries, data types, modules, interfaces, dispinterfaces, and coclasses that are part of the type library. Braces ({}) must surround the definitions.

The following table summarizes the elements that can appear in *definitions*. Each element is described in more detail later in this chapter, in the section “ODL Reference.”

Purpose	Library element	Description
Allows references to other type libraries.	importlib (lib1)	Specifies an external type library that contains definitions that are referenced in this type library.
Declares data types used by the objects in this type library.	typedef [attributes] aliasname	An alias declared using C syntax. Must have at least one attribute to be included in the

type
library.
typedef An
enum enum
[*attributes*] eratio
enum n declar
ed
using
the C
keywo
rds
typedef
enum and
enum.

typedef A
struct structu
[*attributes*] re declar
struct ed
using
the C
keywo
rds
typedef
enum and
struct
.

typedef A
union union
[*attributes*] declar
union ed
using
the C
keywo
rds
typedef
enum and
union.

Descri [*attributes*] Const
bes [*attributes*] ants
functi **module** and
ons **extern** gener
that al data
enabl functio
e ns
queryi whose
ng the action
DLL. s are
not
restrict
ed to

any
specifi
ed
class
of
object
s.

Descri [attrib An
bes utes] interfa
interfa dispint ce
ces. erface descri
bing
the
metho
ds and
proper
ties for
an
object
that
must
be
acces
sed
throug
h
**IDispa
tch::I
nvoke**
.

[attrib An
utes] interfa
interfa ce
ce descri
bing
the
metho
ds and
proper
ties for
an
object
that
can be
acces
sed
either
throug
h
**IDispa
tch::I
nvoke**

or
through
the
VTBL
entries

Describes	[<i>attributes</i>]	Specifies
OLE class	coclass	a top-level object with all of its interfaces and disinterfaces.

In the library description, modules, interfaces, disinterfaces, and coclasses follow the same general syntax:

```
[attributes] elementname typename {
    memberdescriptions
};
```

The *attributes* set characteristics for the element. The *elementname* is a keyword that indicates the kind of item (module, interface, disinterface, or coclass), and the *typename* defines the name of the item. The *memberdescriptions* define the members (constants, functions, properties, and methods) of each element.

Aliases, enumerations, unions, and structures have the following syntax:

```
typedef [typeattributes] typekind typename {
    memberdescriptions
};
```

For these types, the attributes follow the **typedef** keyword, and the *typekind* indicates the data type (**enum**, **union**, or **struct**). For details, see “Attribute Descriptions” later in this chapter.

Note The square brackets ([]) and braces ({ }) in these descriptions are part of the syntax, and are not descriptive symbols. The semicolon after the closing brace (}) that terminates the library definition (and all other type definitions) is optional.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ODL File Example

The following example shows the .odl file for the Lines sample file, extracted from Lines.odl:

```
[
    uuid(3C591B20-1F13-101B-B826-00DD01103DE1),          // LIBID_Lines.
    helpstring("Lines 1.0 Type Library"),
    lcid(0x09),
    version(1.0)
]
library Lines
{
    importlib("stdole.tlb");
    #define DISPID_NEWENUM -4

    [
        uuid(3C591B25-1F13-101B-B826-00DD01103DE1),      // IID_Ipoint.
        helpstring("Point object."),
        oleautomation,
        dual
    ]
    interface IPoint : IDispatch
    {
        [propget, helpstring("Returns and sets x coordinate.")]
        HRESULT x([out, retval] int* retval);
        [propput, helpstring("Returns and sets x coordinate.")]
        HRESULT x([in] int Value);

        [propget, helpstring("Returns and sets y coordinate.")]
        HRESULT y([out, retval] int* retval);
        [propput, helpstring("Returns and sets y coordinate.")]
        HRESULT y([in] int Value);
    }

    // Additional interfaces omitted for brevity.

    [
        uuid(3C591B27-1F13-101B-B826-00DD01103DE1),          // IID_Ipoints.
        helpstring("Points collection."),
        oleautomation,
        dual
    ]
    interface IPoints : IDispatch
    {
        [propget, helpstring("Returns number of points in collection.")]
        HRESULT Count([out, retval] long* retval);

        [propget, id(0),
        helpstring("Given an index, returns a point in the collection.")]
        HRESULT Item([in] long Index, [out, retval] IPoint** retval);

        [propget, restricted, id(DISPID_NEWENUM)]           // Must be propget.
        HRESULT _NewEnum([out, retval] IUnknown** retval);
    }

    // Additional interface omitted for brevity.
```



```

[
    uuid(3C591B22-1F13-101B-B826-00DD01103DE1),    // IID_Iapplication
    helpstring("Application object."),
    oleautomation,
    dual
]
interface IApplication : IDispatch
{
    [propget, helpstring("Returns the application of the object.")]
    HRESULT Application([out, retval] IApplication** retval);

    [propget,
    helpstring("Returns the full name of the application.")]
    HRESULT FullName([out, retval] BSTR* retval);
    [propget, id(0),
    helpstring("Returns the name of the application.")]
    HRESULT Name([out, retval] BSTR* retval);

    [propget, helpstring("Returns the parent of the object.")]
    HRESULT Parent([out, retval] IApplication** retval);

    [propput]
    HRESULT Visible([in] boolean VisibleFlag);
    [propget, helpstring
    ("Sets or returns whether the main window is visible.")]
    HRESULT Visible([out, retval] boolean* retval);

    [helpstring("Exits the application.")]
    HRESULT Quit();

// Additional methods omitted for brevity.

    [helpstring("Creates new Point object initialized to (0,0).")]
    HRESULT CreatePoint([out, retval] IPoint** retval);
}

[
    uuid(3C591B21-1F13-101B-B826-00DD01103DE1),    // CLSID_Lines.
    helpstring("Lines Class"),
    appobject
]
coclass Lines
{
    [default] interface IApplication;
    interface IDispatch;
}
}

```

The example describes a library named Lines that imports the standard OLE library Stdole.tlb. The **#define** directive defines the constant DISPID_NEWENUM, which is needed for the **_NewEnum** property of the **IPoints** collection. (Note that the use of preprocessor directives requires that you have the C preprocessor installed on your machine.)

The example shows declarations for three interfaces in the library: **IPoint**, **IPoints**, and **IApplication**. Because all three are dual interfaces, their members can be invoked through **IDispatch** or directly through VTBLs. In addition, all of their members return HRESULT values and pass their return values as **retval** parameters. Therefore, they can support the **IErrorInfo** interface, through which they can return detailed error information in whatever way they are invoked.

The **IPoint** interface has two properties, **X** and **Y**, and two pairs of accessor functions to get and set the properties.

The **IPoint** interface is a collection of points. It supports three read-only properties, each of which has a single accessor function. The **Count** and **Item** properties return the number of points and the value of a single point, respectively. The **_NewEnum** property, required for collection objects, returns an enumerator object for the collection. This property has the **restricted** attribute, indicating that it should not be invoked from a macro language.

The **IApplication** interface describes the application object. It supports the properties **Application**, **FullName**, **Name**, **Parent**, **Visible**, and **Pane**. It supports the methods **Quit**, **CreateLine**, and **CreatePoint**.

Finally, the script defines a coclass named Lines. The **appobject** attribute makes the members of the coclass (**IApplication** and **IDispatch**) globally accessible in the type library. **IApplication** is defined as the **default** member, indicating that it is the programmability interface intended for use by macro languages.

Source File Contents

The following sections describe the proper format for comments, constants, identifiers, and other syntactic items in an .odl file.

Array Definitions

MkTypLib accepts both fixed-size arrays and arrays declared as SAFEARRAY.

Use a C-style syntax for a fixed size array:

```
type arrname[size];
```

To describe a SAFEARRAY, use the following syntax:

```
SAFEARRAY (elementtype) *arrayname
```

A function returning a SAFEARRAY has the following syntax:

```
SAFEARRAY (elementtype) myfunction(parameterlist);
```

Comments

To include comments in an .odl file, use a C-style syntax in either block form (*/*...*/*) or single-line form (*//*). MkTypLib ignores the comments, and does not preserve them in the header (.h) file.

Constants

A constant can be either numeric or a string, depending on the attribute.

Numeric

Numeric input is usually an integer (in either decimal or in hexadecimal, using the standard 0x format), but can also be a single character constant (for example, \0).

String

A string is delimited by double quotation marks (") and cannot span multiple lines. The backslash character (\) acts as an escape character. The backslash character followed by any character (even another backslash) prevents the second character from being interpreted with any special meaning.

The backslash is not included in the text.

For example, to include a double quotation mark (") in the text without causing it to be interpreted as the closing delimiter, it should be preceded with a backslash (\"). Similarly, a double backslash (\\) should be used to put a backslash into the text. Some examples of valid strings are:

```
"commandName"  
"This string contains a \"quote\"."  
"Here's a pathname: c:\\bin\\binp"
```

A string can be up to 255 characters long.

File Names

A file name is a string that represents either a full or partial path. OLE Automation expects to find files in directories that are referenced by the type library registration entries, so partial path names are typically used. For more information about registration, refer to Chapter 2, “Exposing OLE Automation Objects.”

Forward Declarations

Forward declarations permit forward references to types. Forward references have the following form:

```
typedef struct mydata;  
interface aninterface;  
dispinterface fordipatch;  
coclass pococlass;
```

Globally Unique ID (GUID)

A universally unique ID (UUID) is a globally unique ID (GUID). This number is created by running the Guidgen.exe command line program. Guidgen.exe never produces the same number twice, no matter how many times it is run or how many different machines it runs on. Every entity that needs to be uniquely identified (such as an interface) has a globally unique ID.

Identifiers

Identifiers can be up to 255 characters long, and must conform to C-style syntax. MkTypLib is case sensitive, but it generates type libraries that are case insensitive. It is therefore possible to define a user-defined type whose name differs from that of a built-in type only by case. User-defined type names (and member names) that differ only in case refer to the same type or member. Except for property accessor functions, it is invalid for two members of a type to have the same name, regardless of case.

Intrinsic Data Types

The following data types are recognized by MkTypLib:

Type	Description
boolean	Data item that can have the value True or False. The size maps to VARIANT_BOOL

OOL.

char	8-bit signed data item.
double	64-bit IEEE floating-point number.
int	Signed integer, whose size is system dependent.
float	32-bit IEEE floating point number.
long	32-bit signed integer.
short	16-bit signed integer.
wchar _t	Unicode character accepted only for 32-bit type libraries.
BSTR	Length-prefixed string, as described in Chapter 5, "Dispatch Interface and API Functions."
CURR ENCY	8-byte, fixed-point number.
DATE	64-bit floating-point fractional number of days since December 30, 1899.
SCOD E	Built-in error type that corresponds to VT_ERROR. An SCODE does not contain the additional error

information
provided by
HRESULT.

VARIANT One of the
variant data
types as
described in
Chapter 5,
“Dispatch
Interface
and API
Functions.”

IDispatch * Pointer to
the
IDispatch
interface.

IUnknown * Pointer to
the
IUnknown
interface.
(Any OLE
interface can
be
represented
by its
IUnknown
interface.)

SAFEARRAYTYPEOF(Typ
eName)
TypeName
is any of the
above types.
Array of
these types.

TypeName*
is any of the
above types.
Pointer to a
type.

void Allowed only
as return
type for a
function, or
in a function
parameter
list to
indicate no
arguments.

HRESULT Return type
used for
reporting
error
information
in interfaces,
as described
in *Microsoft*

*OLE
Programmer's
Guide and
Reference.*

LPWSTR	Unicode string accepted only for 32-bit type libraries.
LPSTR	Zero-terminated string.

Not all of the above types can be marshaled by OLE Automation to another process or thread. The list of types that can be marshaled are called OLE Automation compatible types, and are listed under the **oleautomation** attribute description.

The keyword **unsigned** can be specified before **int**, **char**, **short**, and **long**.

String Definitions

Strings can be declared using the LPSTR data type, which indicates a zero-terminated string, and with the BSTR data type, which indicates a length-prefixed string (as defined in Chapter 5, "Dispatch Interface and API Functions"). In 32-bit type libraries, Unicode strings can be defined with the LPWSTR data type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ODL Reference

This section provides reference material on the attributes, statements, and directives that are part of the Object Description Language (ODL).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Attribute Descriptions

The following sections describe the ODL attributes and the types of objects that they apply to, along with the equivalent flags set in the object's type information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


appobject

Description

Identifies the Application object.

Allowed on

Coclass.

Comments

Indicates that the members of the class can be accessed without qualification when accessing this type library.

Flags

TYPEFLAG_FAPPOBJECT

{ewl msdncd.dll, ewcright, /c"Microsoft"}

aggregatable

Description

Indicates that the class supports aggregation.

Allowed on

Coclass.

Comments

Indicates that the members of the class can be accessed without qualification when accessing this type library.

Flags

TYPEFLAG_FAGGREGATABLE

Example

```
[ uuid(1e196b20-1f3c-1069-996b-00dd010fe676),  
  aggregatable  
]  
coclass Form  
{  
    [default] interface IForm;  
    [default, source] interface IFormEvents;  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

bindable

Description

Indicates that the property supports data binding.

Allowed on

Property.

Comments

Refers to the property as a whole, so it must be specified wherever the property is defined. The attribute should be specified on both the property get description and the property set description.

Flags

FUNCFLAG_FBINDABLE, VARFLAG_FBINDABLE

{ewl msdncd.dll, ewcright, /c"Microsoft"}

control

Description

Indicates that the item represents a control from which a container site will derive additional type libraries or coclasses.

Allowed on

Type libraries, coclasses.

Comments

This attribute allows type libraries that describe controls to be marked so that they are not be displayed in type browsers intended for nonvisual objects.

Flags

TYPEFLAG_FCONTROL, LIBFLAG_FCONTROL

{ewl msdncd.dll, ewcright, /c"Microsoft"}

custom(<guid>, <value>)

Description

Indicates a custom attribute (one not defined by OLE Automation). This feature enables the independent definition and use of attributes.

<g the standard
ui GUID form
d>
<v a value that can
al be put into a
ue variant. (See
> also the **Const**
directive.)

Allowed on

Library, type information, variable, function, parameter.

Not Allowed on

A member of a coclass.

Representation

Can be retrieved using:

ITypelib2::GetCustData
TypeInfo2::GetCustData
ITypelib2::GetFuncCustData
ITypelib2::GetVarCustData
ITypelib2::GetParamCustData

Example

The following example shows how to add a string-valued attribute that gives the programmatic ID (ProgID) for a class:

```
[
    custom(GUID_PROGID, "DAO.Dynaset")
]
coclass Dynaset
{
    [default] interface Dynaset;
    [default, source] interface IDynasetEvents;
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

default

Description

Indicates that the interface or dispinterface represents the default programmability interface. Intended for use by macro languages.

Allowed on

Coclass member.

Comments

A coclass can have two **default** members at most. One represents the source interface or dispinterface, and the other represents the sink interface or dispinterface. If the **default** attribute is not specified for any member of the coclass or cotype, the first source and sink members that do not have the **restricted** attribute will be treated as the defaults.

Flags

IMPLTYPEFLAG_FDEFAULT

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


defaultbind

Description

Indicates the single, bindable property that best represents the object.

Allowed on

Property.

Comments

Properties that have the **defaultbind** attribute must also have the **bindable** attribute. The **defaultbind** attribute cannot be specified on more than one property in a dispinterface.

This attribute is used by containers that have a user model that involves binding to an object rather than binding to a property of an object. An object can support data binding and not have this attribute.

Flags

FUNCFLAG_FDEFAULTBIND, VARFLAG_FDEFAULTBIND

{ewl msdncd.dll, ewcright, /c"Microsoft"}

defaultcollelem

Description

Allows for optimization of code.

Allowed on

Type information, property.

Comments

In VBA, "foo!bar" is normally syntactic shorthand for foo.defaultprop("bar"). Because such a call is significantly slower than accessing a data member of foo directly, an optimization has been added in which the compiler looks for a member named "bar" on the type of foo. If such a member is found and flagged as an accessor function for an element of the default collection, a call is generated to that member function. To allow vendors to produce object servers that will be optimized in this way, the member flag should be documented.

Because this optimization searches the type of item that precedes the '!', it will optimize calls of the form MyForm!bar only if MyForm has a member named "bar," and it will optimize MyForm.Controls!bar only if the return type of Controls has a member named **bar**. Even though MyForm!bar and MyForm.Controls!bar both would normally generate the same calls to the object server, optimizing these two forms requires that the object server add the **bar** method in both places.

Use of [defaultcolitem] must be consistent for a property. For example, if it is present on a **Get**, it must also be present on a **Put**.

Flags

FUNCFLAG_FDEFAULTCOLLELEM, VARFLAG_FDEFAULTCOLLELEM

Example

A form has a button on it named **Button1**. User code can access the button using property syntax or ! syntax, as shown below.

```
Sub Test()  
    Dim f As Form1  
    Dim b1 As Button  
    Dim b2 As Button  
  
    Set f = Form1  
  
    Set b1 = f.Button1      ' Property syntax  
    Set b = f!Button1      ' ! syntax  
End Sub
```

To use the property syntax and the ! syntax properly, see the form in the type information below.

```
[ odl,  
  dual,  
  uuid(1e196b20-1f3c-1096-996b-00dd010ef676),  
  helpstring("This is IForm"),  
  restricted  
]  
interface IForm1: IForm  
{
```



```
[propget, defaultcollelem]  
HRESULT Button1([out, retval] Button *Value);  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


defaultnext [TBD]

Description

???

Allowed on

???

Comments

???

???

Flags

???

{ewl msdncd.dll, ewcright, /c"Microsoft"}

defaultvalue(value)

Description

Enables specification of a default value for a typed optional parameter.

Allowed on

Parameter.

Comments

The expression *value* resolves to a constant that can be described in a variant. The ODL already allows some expression forms, as when a constant is declared in a module. The same expressions are supported without modification.

The following example shows some legal parameter descriptions:

```
interface IFoo
{
    void Ex1([defaultvalue(44)] LONG        i);
    void Ex2([defaultvalue(44)] SHORT       i);
    void Ex3([defaultvalue("Hello")] BSTR    i);
}
```

The following rules apply:

1. It is invalid to specify a default value for a parameter whose type is a safe array. It is invalid to specify a default value for any type that cannot go in a variant, including structures and arrays.
2. Optional parameters must follow default value parameters. Optional parameters and default value parameters must follow mandatory parameters.
3. The default value can be any constant that is represented by a variant.

Flags

[TBD]

Example

```
interface QueryDef
{
    // Type is now known to be a LONG type (good for browser in VBA and
    // for a C/C++ programmer) and also has a default value of
    // dbOpenTable (constant).

    HRESULT OpenRecordset(          [in, defaultvalue(dbOpenTable)]
        LONG Type,

        [out,retval]
        Recordset **pprst);
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

defaultvtbl

Description

Enables an object to have two different source interfaces.

The **default** interface is an interface or dispinterface that is the default source interface. If the interface is a:

- **dual** interface, sinks receive events through **IDispatch**.
- **VTBL** interface, sinks receive events through VTBL.
- **dispinterface**, sinks receive events through **IDispatch**.
- **defaultvtable**, a default VTBL interface, which cannot be a dispinterface — it must be a dual, VTBL, or interface. If the interface is a dual interface, then sinks receive events through the VTBL.

An object can have both a default source and a default VTBL source interface with the same interface ID. In this case, a sink should advise using IID_IDISPATCH to receive dispatch events, and use the specific interface ID to receive VTBL events.

Allowed on

A member of a coclass.

Comments

For normal (non-source) interfaces, an object can support a single interface that satisfies consumers who want to use **IDispatch** access as well as VTBL access (a dual interface). Because of the way source interfaces work, it is not possible to do the same for source interfaces. The object with the source interface is in control of whether calls are made through **IDispatch** or through the VTBL. The object has to do one or the other, and without changing something, the object will not know which to use. The sink does not provide any information about how it wants to receive the events. The only action that object-sourcing events can take would be to use the least common denominator, the **IDispatch** interface. This effectively reduces a dual interface to a dispatch interface with regard to sourcing events.

Interf ace	Flag it translates into
defau lt	IMPLTYPEFL AG_FDEFAU LT
defau lt, sour ce	IMPLTYPEFL AG_FDEFAU LT IMPLTYPEFL AG_FSOUR CE
defau ltvtab le, sour ce	IMPLTYPEFL AG_FDEFAU LT

IMPLTYPEFL
 AG_FDEFAU
 LTVTABLE
 IMPLTYPEFL
 AG_FSOURL
 CE

Flags

IMPLTYPEFLAG_FDEFAULTVTABLE. If this flag is set, then IMPLTYPEFLAG_FDEFAULT is also set.

Example

```
[ odl,
  dual,
  uuid(1e196b20-1f3c-1069-996b-00dd010ef676),
  restricted
]
interface IForm: IDispatch
{
    [propget]
    HRESULT Backcolor([out, retval] long *Value);

    [propput]
    HRESULT Backcolor([in] long Value);

    [propget]
    HRESULT Name([out, retval] BSTR *Value);

    [propput]
    HRESULT Name([in] BSTR Value);
}

[ odl,
  dual,
  uuid(1e196b20-1f3c-1069-996b-00dd010ef767),
  restricted
]
interface IFormEvents: IDispatch
{
    HRESULT Click();
    HRESULT Resize();
}

[uuid(1e196b20-1f3c-1069-996b-00dd010fe676)]
coclass Form
{
    [default] interface IForm;
    [default, source] interface IFormEvents;
    [defaultvttable, source] interface IFormEvents;
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

displaybind

Description

Indicates that a property should be displayed as bindable to the user.

Allowed on

Property.

Comments

Properties that have the **displaybind** attribute must also have the **bindable** attribute. An object can support data binding and not have this attribute.

Flags

FUNCFLAG_FDISPLAYBIND, VARFLAG_FDISPLAYBIND

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


`dllname(str)`

Description

Defines the name of the DLL that contains the entry points for a module.

Allowed on

Module (required).

Comments

The *str* argument gives the file name of the DLL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


dual

Description

Identifies an interface that exposes properties and methods through **IDispatch** and directly through the VTBL.

Allowed on

Interface.

Comments

The interface must be compatible with OLE Automation and derive from **IDispatch**. Not allowed on dispinterfaces.

The **dual** attribute creates an interface that is both a **Dispatch** interface and a Component Object Model (COM) interface. The first seven entries of the VTBL for a dual interface are the seven members of **IDispatch**, and the remaining entries are COM entries for direct access to members of the dual interface. All of the parameters and return types specified for members of a dual interface must be compatible with OLE Automation types.

All members of a dual interface must pass an HRESULT as the function's return value. Members that need to return other values should specify the last parameter as **[retval, out]** indicating an output parameter that returns the value of the function. In addition, members that need to support multiple locales should pass an *lcid* parameter.

A dual interface provides for both the speed of direct VTBL binding and the flexibility of **IDispatch** binding. For this reason, dual interfaces are recommended whenever possible.

Note If an application accesses object data by casting the THIS pointer in the interface call, the VTBL pointers in the object should be checked against the VTBL pointers to ensure that they are connected to the appropriate proxy.

Specifying dual on an interface implies that the interface is compatible with OLE Automation, and therefore causes both the TYPEFLAG_FDUAL and TYPEFLAG_FOLEAUTOMATION flags to be set.

Flags

TYPEFLAG_FDUAL, TYPEFLAG_FOLEAUTOMATION

{ewl msdncd.dll, ewcright, /c"Microsoft"}

entry(entryid)

Description

Identifies the entry point in the DLL.

Allowed on

Functions in a module (required).

Comments

If *entryid* is a string, this is a named entry point. If *entryid* is a number, the entry point is defined by an ordinal. This attribute provides a way to obtain the address of a function in a module.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


helpcontext(numctxt)

Description

Sets the context in the Help file.

Allowed on

Library, interface, dispinterface, struct, enum, union, module, typedef, method, struct member, enum value, property, coclass, const.

Comments

Retrieved by the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces. The *numctxt* is a 32-bit Help context ID in the Help file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


helpfile(filename)

Description

Sets the name of the Help file.

Allowed on

Library.

Comments

Retrieved through the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces.

All types in a library share the same Help file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

helpstring(string)

Description

Sets the Help string.

Allowed on

Library, interface, dispinterface, struct, enum, union, module, typedef, method, struct member, enum value, property, coclass, const.

Comments

Retrieved through the **GetDocumentation** functions in the **ITypeLib** and **ITypeInfo** interfaces.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


helpstringcontext(contextid)

Description

Sets the string context in the Help file.

Allowed on

Type library, type information (TypeInfo), function, and variable level.

Comments

Retrieved by the **GetDocumentation2** functions in the **ITypeLib2** and **ITypeInfo2** interfaces. The *contextid* is a 32-bit Help context ID in the Help file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


helpstringdll(dllname)

Description

Sets the name of the DLL to use to perform the doc string lookup (localization).

Allowed on

Type library.

Comments

Retrieved through the **GetDocumentation2** functions in the **ITypeLib2** and **TypeInfo2** interfaces.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

hidden

Description

Indicates that the item exists, but should not be displayed in a user-oriented browser.

Allowed on

Property, method, coclass, dispinterface, interface, library.

Comments

This attribute allows members to be removed from an interface by shielding them from further use, while maintaining compatibility with existing code.

When specified for a library, the attribute prevents the entire library from being displayed. It is intended for use by controls. Hosts need to create a new type library that wraps the control with extended properties.

Flags

VARFLAG_FHIDDEN, FUNCFLAG_FHIDDEN, TYPEFLAG_FHIDDEN

{ewl msdncd.dll, ewcright, /c"Microsoft"}

id(num)

Description

Identifies the DISPID of the member.

Allowed on

Method or property in an interface or dispinterface.

Comments

The *num* is a 32-bit integral value in the following format:

Bits	Value
0	Offset. Any
–	value is
15	permissible.
16	The nesting
–	level of this
21	TypeInfo in the inheritance
	heirarchy. For example:
	interface mydisp
	: IDispatch
	The nesting
	level of
	Unknown is 0,
	IDispatch is 1,
	and MyDisp is 2.
22	Reserved. Must
–	be zero.
25	
26	DISPID value.
–	
28	
29	True if this is the member ID for a
	FuncDesc;
	otherwise False.
30	Must be 01.
–	
31	

Negative IDs are reserved for use by OLE Automation.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

immediatebind

Description

Allows individual bindable properties on a form to specify this behavior. When this bit is set, all changes will be notified.

Allowed on

Flags.Comments

Allows controls to differentiate two different types of bindable properties. One type of bindable property needs to notify every change to the database (for example, with a check box control where every change needs to be sent through to the underlying database, even though the control has not lost the focus). However, controls such as a list box need to have the change of a property communicated to the database when the control loses focus, because the user may have changed the selection with the arrow keys before finding the desired setting. If the change notification was sent to the database every time the user pressed an arrow key, it would give an unacceptable performance.

VARFLAG_FIMMEDIATEBIND
FUNCFLAG_FIMMEDIATEBIND

The **bindable** and **requestedit** attribute bits need to be set for this new bit to have an effect.

Description

Specifies an input parameter.

Allowed on

Parameter.

Comments

The parameter can be a pointer (such as *char**) but the value it refers to is not returned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Lcid

Description

Indicates that the parameter is a locale ID.

Allowed on

Parameter in a member of an interface.

Comments

Only one parameter can have this attribute. The parameter must have the *in* attribute and not the **out** attribute, and its type must be **long**. The **lcid** attribute is not allowed on dispinterfaces.

The **lcid** attribute allows members in the VTBL to receive an LCID at the time of invocation. By convention, the **lcid** parameter is the last parameter not to have the **retval** attribute. If the member specifies *propertyput* or *propertyputref*, the **lcid** parameter must precede the parameter that represents the right side of the property assignment.

TypeInfo::Invoke passes the LCID of the type information into the **lcid** parameter. Parameters with this attribute are not displayed in user-oriented browsers.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


lcid(numid)

Description

Identifies the locale for a type library.

Allowed on

Library.

Comments

The *numid* is a 32-bit locale ID, as used in Win32 National Language Support. The locale ID is typically entered in hexadecimal format.

```
{ewl msdncl.dll, ewcright, /c"Microsoft"}
```


licensed

Description

Indicates that the class is licensed.

Allowed on

Coclass.

Flags

TYPEFLAG_FLICENSED

{ewl msdncd.dll, ewcright, /c"Microsoft"}

nonbrowsable

Description

Indicates that the property appears in an object browser (which does not show property values), but does not appear in a properties browser (which does show property values).

Allowed on

Property.

Flags

VARFLAG_FNONBROWSABLE, FUNCFLAG_FNONBROWSABLE

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


noncreatable

Description

Indicates that the class does not support creation at the top level (for example, through **TypeInfo::CreateInstance** or **CoCreateInstance**). An object of such a class is usually obtained through a method call on another object.

Allowed on

Coclass.

Flags

TYPEFLAG_FCANCREATE

Example

```
[
    uuid(1e196b20-1fc3-1069-996b-00dd010ef671),
    helpstring("This is Dynaset"),
    noncreatable
]
coclass Dynaset
{
    [default] interface IDynaset;
    [default, source] interface IDynasetEvents;
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

nonextensible

Description

Indicates that the **IDispatch** implementation includes only the properties and methods listed in the interface description.

Allowed on

Dispinterface, interface.

Comments

The interface must have the **dual** attribute.

By default, OLE Automation assumes that interfaces can add members at run time, meaning that it assumes the interfaces are extensible.

Flags

TYPEFLAG_FNONEXTENSIBLE

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


odl

Description

Identifies an interface as an Object Description Language (ODL) interface.

Allowed on

Interface (required).

Comments

This attribute must appear on all interfaces.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

oleautomation

Description

The **oleautomation** attribute indicates that an interface is compatible with OLE Automation.

Allowed on

Interface.

Comments

Not allowed on dispinterfaces.

The parameters and return types specified for its members must be compatible with OLE Automation, as listed in the following table:

Type	Description
boolean	Data item that can have the value True or False. The size corresponds to VARIANT_BOOL. Use VT_TRUE, VT_FALSE.
unsigned char	8-bit unsigned data item.
double	64-bit IEEE floating-point number.
float	32-bit IEEE floating-point number.
int	Signed integer, whose size is system dependent.

	t.
long	32-bit signed integer.
short	16-bit signed integer.
BSTR	Length-prefixed string, as described in Chapter 5, "Dispatch Interface and API Functions."
CURRENCY	8-byte, fixed-point number.
DATE	64-bit, floating-point fractional number of days since December 30, 1899.
SCODE	Built-in error type that corresponds to VT_ERROR.
typedef enum <i>myenum</i>	Signed integer, whose size is system dependent.
interface IDispatch *	Pointer to the IDispatch interface (VT_DISPATCH).
interface	Pointer to

IUnknown n *	an interface that does not derive from IDispatch (VT_UNK NOWN). (Any OLE interface can be represent ed by its IUnknown interface.)
dispinterf ace <i>Typenam</i> e *	Pointer to an interface derived from IDispatch (VT_DISP ATCH).
coclass <i>Typenam</i> e *	Pointer to a coclass name (VT_UNK NOWN).
[oleauto mation] interface <i>Typenam</i> e *	Pointer to an interface that derives from IDispatch.
SAFEARR RAY(<i>Typ</i> <i>eName</i>)	<i>TypeNam</i> e is any of the above types. Array of these types.
TypeNam e*	<i>TypeNam</i> e is any of the above types. Pointer to a type.

A parameter is compatible with OLE Automation if its type is compatible with an OLE Automation type, a pointer to an OLE Automation type, or a SAFEARRAY of an OLE Automation type.

A return type is compatible with OLE Automation if its type is an HRESULT or is **void**. Methods in OLE Automation must return either HRESULT or **void**.

A member is compatible with OLE Automation if its return type and all of its parameters are compatible with OLE Automation.

An interface is compatible with OLE Automation if it derives from **IDispatch** or **IUnknown**, if it has the **oleautomation** attribute, or if all of its VTBL entries are compatible with OLE Automation. For 32-bit systems, the calling convention for all methods in the interface must be STDCALL. For 16-bit systems, all methods must have the CDECL calling convention. Every dispinterface is compatible with OLE Automation.

Flags

TYPEFLAG_FOLEAUTOMATION

{ewl msdncd.dll, ewcright, /c"Microsoft"}

optional

Description

Specifies an optional parameter.

Allowed on

Parameter.

Comments

Valid only if the parameter is of type VARIANT or VARIANT*. All subsequent parameters of the function must also be **optional**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


out

Description

Specifies an output parameter.

Allowed on

Parameter.

Comments

The parameter must be a pointer to memory that will receive a result.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


propget

Description

Specifies a property-accessor function.

Allowed on

Functions, methods in interfaces, dispinterfaces.

Comments

The property must have the same name as the function. At most, one of **propget**, **propput**, and **propputref** can be specified for a function.

Flags

INVOKE_PROPERTYGET

{ewl msdncd.dll, ewcright, /c"Microsoft"}

propput

Description

Specifies a property-setting function.

Allowed on

Functions, methods in interfaces, dispinterfaces.

Comments

The property must have the same name as the function. Only one **propget**, **propput**, and **propputref** can be specified.

Flags

INVOKE_PROPERTYPUT

{ewl msdncd.dll, ewcright, /c"Microsoft"}

propputref

Description

Specifies a property-setting function that uses a reference instead of a value.

Allowed on

Functions, methods in interfaces, dispinterfaces.

Comments

The property must have the same name as the function. Only one **propget**, **propput**, and **propputref** can be specified.

Flags

INVOKE_PROPERTYPUTREF

{ewl msdncd.dll, ewcright, /c"Microsoft"}

public

Description

Includes an alias declared with the **typedef** keyword in the type library.

Allowed on

Alias declared with **typedef**.

Comments

By default, an alias that is declared with **typedef**, and has no other attributes, is treated as a **#define**, and is not included in the type library. Using the **public** attribute ensures that the alias becomes part of the type library.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


readonly

Description

Prohibits assignment to a variable.

Allowed on

Variable.

Flags

VARFLAG_FREADONLY

{ewl msdncd.dll, ewcright, /c"Microsoft"}

replaceable

Description

Tags an interface as having default behaviors.

Allowed on

Coclass, variable.

Comments

The object supports **IConnectionPointWithDefault**.

Flags

TYPEFLAG_FREPLACEABLE, FUNCFLAG_FREPLACEABLE, VARFLAG_FREPLACEABLE

{ewl msdncd.dll, ewcright, /c"Microsoft"}

requestededit

Description

Indicates that the property supports the **OnRequestEdit** notification.

Allowed on

Property.

Comments

The property supports the **OnRequestEdit** notification, raised by a property before it is edited. An object can support data binding and not have this attribute.

Flags

FUNCFLAG_FREQUESTEDIT, VARFLAG_FREQUESTEDIT

{ewl msdncd.dll, ewcright, /c"Microsoft"}

restricted

Description

Prevents the item from being used by a macro programmer.

Allowed on

Type library, type information, coclass member, or member of a module or interface.

Comments

This attribute is allowed on a member of a coclass, independent of whether the member is a dispinterface or interface, and independent of whether the member is a sink or source. A member of a coclass cannot have both the **restricted** and **default** attributes.

Flags

IMPLTYPEFLAG_FREstricted, FUNCFLAG_FREstricted

```
[ odl,
    dual,
    uuid(1e196b20-1f3c-1069-996b-00dd010ef676),
    helpstring("This is IForm"),
    restricted
]
interface IForm: IDispatch
{
    [propget]
    HRESULT Backcolor([out, retval] long *Value);

    [propput]
    HRESULT Backcolor([in] long Value);
}

[ odl,
    dual,
    uuid(1e196b20-1f3c-1069-996b-00dd010ef767),
    helpstring("This is IFormEvents"),
    restricted
]
interface IFormEvents: IDispatch
{
    HRESULT Click();
}

[ uuid(1e196b20-1f3c-1069-996b-00dd010fe676),
  helpstring("This is Form")
]
coclass Form
{
    [default] interface IForm;
    [default, source] interface IFormEvents;
}
```



```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


retval

Description

Designates the parameter that receives the return value of the member.

Allowed on

Parameters of interface members that describe methods or get properties.

Comments

This attribute can be used only on the last parameter of the member. The parameter must have the **out** attribute and must be a pointer type.

Parameters with this attribute are not displayed in user-oriented browsers.

Flags

IDLFLAG_FRETVAL

{ewl msdncd.dll, ewcright, /c"Microsoft"}

source

Description

Indicates that a member is a source of events.

Allowed on

Member of a coclass, property, or method.

Comments

For a member of a coclass, this attribute indicates that the member is called rather than implemented.

On a property or method, this attribute indicates that the member returns an object or VARIANT that is a source of events. The object implements the interface **IConnectionPointContainer**.

Flags

IMPLTYPEFLAG_FSOURCE, VARFLAG_SOURCE, FUNCFLAG_SOURCE

{ewl msdncd.dll, ewcright, /c"Microsoft"}

string

Description

Specifies a string.

Allowed on

Structure, member, parameter, property.

Comments

Included only for compatibility with the Interface Definition Language (IDL). Use LPSTR for a zero-terminated string.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


uidefault

Description

Indicates that the type information member is the default member for display in the user interface.

Allowed on

A member of an interface or dispinterface.

Comments

This attribute is used to mark an event as the default (the first one created) or a property as the default (the one to select first in the properties browser).

For example, Visual Basic uses this attribute in the following ways:

- When an object is double-clicked at design time, Visual Basic jumps to the event in the default source interface that is marked as **[uidefault]**. If there is no such member, then Visual Basic displays the first one listed in the default source interface.
- When an object is selected at design time, by default, the Properties window in Visual Basic displays the property in the default interface that is marked as **[uidefault]**. If there is no such member, then Visual Basic displays the first one listed in the default interface.

Flags

FUNCFLAG_FUIDEFAULT, VARFLAG_FUIDEFAULT

```
[ odl,
  dual,
  uuid(1e196b20-1f3c-1069-996b-00dd010ef676),
  restricted
]
interface IForm: IDispatch
{
    [propget]
    HRESULT Backcolor([out, retval] long *Value);

    [propput]
    HRESULT Backcolor([in] long Value);

    [propget, uidefault]
    HRESULT Name([out, retval] BSTR *Value);

    [propput, uidefault]
    HRESULT Name([in] BSTR Value);
}

[ odl,
  dual,
  uuid(1e196b20-1f3c-1069-996b-00dd010ef767),
  restricted
]
interface IFormEvents: IDispatch
{
    [uidefault]
```



```
        HRESULT Click();

        HRESULT Resize();
    }

    [uuid(1e196b20-1f3c-1069-996b-00dd010fe676)]
    coclass Form
    {
        [default] interface IForm;
        [default, source] interface IFormEvents;
    }
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


uuid(uuidval)

Description

Specifies the universally unique ID (UUID) of the item.

Allowed on

Required for library, dispinterface, interface, and coclass. Optional for struct, enum, union, module, and typedef.

Comments

The **uuidval** is a 16-byte value using hexadecimal digits in the following format: 12345678-1234-1234-1234-123456789ABC. This value is returned in the **TypeAttr** structure retrieved by **TypeInfo::GetTypeAttr**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


vararg

Description

Indicates a variable number of arguments.

Allowed on

Function.

Comments

Indicates that the last parameter is a safe array of VARIANT type, which contains all of the remaining parameters.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


version(versionval)

Description

Specifies a version number.

Allowed on

Library, struct, module, dispinterface, interface, coclass, enum, union.

Comments

The argument *versionval* is a real number in the format *n.m*, where *n* is a major version number and *m* is a minor version number.

ODL Statements and Directives

The following sections describe the statements and directives that make up the Object Description Language (ODL).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


coclass Statement

Describes the globally unique ID (GUID) and the supported interfaces for a Component Object Model (COM).

Syntax

```
[attributes]
    coclass classname {
        [attributes2] [interface | dispinterface] interfacename;
        // Code omitted here for brevity.
    };
```

Syntax Elements

attributes

The **uuid** attribute is required on a coclass. This is the same **uuid** that is registered as a CLSID in the system registration database. The **helpstring**, **helpcontext**, **licensed**, **version**, **control**, **hidden**, and **appobject** attributes are accepted, but not required, before a coclass definition. For more information about the attributes accepted before a coclass definition, see “Attribute Descriptions” earlier in this chapter. The **appobject** attribute makes the functions and properties of the coclass globally available in the type library.

classname

Name by which the common object is known in the type library.

attributes2

Optional attributes for the interface or dispinterface. The **source**, **default**, and **restricted** attributes are accepted on an interface or dispinterface in a coclass.

interfacename

Either an interface declared with the **interface** keyword, or a dispinterface declared with the **dispinterface** keyword.

Comments

The Component Object Model defines a class as an implementation that allows **QueryInterface** between a set of interfaces.

Example

```
[ uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0), helpstring("A
class"), helpcontext(2481), appobject]
coclass myapp {
    [source] interface IMydocfuncs;
    dispinterface DMydocfuncs;
};

[uuid 00000000-0000-0000-0000-123456789019]
coclass fooLeslie Brown
{
    [restricted] interface bar;
    interface bar;
}
```



```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


dispinterface Statement

Defines a set of properties and methods on which **IDispatch::Invoke** can be called. A dispinterface can be defined by explicitly listing the set of supported methods and properties (Syntax 1) or by listing a single interface (Syntax 2).

Syntax 1

```
[attributes]
    dispinterface intfname {
        properties:
            proplist
        methods:
            methlist
    };
```

Syntax 2

```
[attributes]
    dispinterface intfname {
        interface interfacename
    };
```

Syntax Elements

attributes

The **helpstring**, **helpcontext**, **hidden**, **uuid**, and **version** attributes are accepted before **dispinterface**. For more information about the attributes accepted before a **dispinterface** definition, see “Attribute Descriptions” earlier in this chapter. Attributes (including the brackets) can be omitted, except for the **uuid** attribute, which is required.

intfname

The name by which the dispinterface is known in the type library. This name must be unique within the type library.

interfacename

(Syntax 2) The name of the interface to declare as an **IDispatch** interface.

proplist

(Syntax 1) An optional list of properties supported by the object, declared in the form of variables. This is the short form for declaring the property functions in the methods list. See the comments section for details.

methlist

(Syntax 1) A list comprising a function prototype for each method and property in the dispinterface. Any number of function definitions can appear in *methlist*. A function in *methlist* has the following form:

```
[attributes] returntype methname(params);
```

The following attributes are accepted on a method in a dispinterface: **helpstring**, **helpcontext**, **string** (for compatibility with the Interface Definition Language), **bindable**, **defaultbind**, **displaybind**, **propget**, **propput**, **propputref**, and **vararg**. If **vararg** is specified, the last parameter must be a safe array of VARIANT type.

The parameter list is a comma-delimited list, each element of which has the following form:

```
[attributes] type paramname
```


The *type* can be any declared or built-in type, or a pointer to any type. Attributes on parameters are **in**, **out**, **optional**, and **string**.

If **optional** is specified, it must only be specified on the right-most parameters, and the types of those parameters must be VARIANT.

Comments

Method functions are specified exactly as described in “module Statement,” except that the **entry** attribute is not allowed.

Note Stdole32.tlb (Stdole.tlb on 16-bit systems) must be imported, because a dispinterface inherits from **IDispatch**.

Properties can be declared either in the properties or methods lists. Declaring properties in the properties list does not indicate the type of access the property supports (**get**, **put**, or **putref**). Specify the **readonly** attribute for properties that do not support **put** or **putref**. If the property functions are declared in the methods list, functions for one property will all have the same ID.

Using Syntax 1, the *properties:* and *methods:* tags are required. The **id** attribute is also required on each member. For example:

```
properties:
    [id(0)] int Value; // Default property.
methods:
    [id(1)] void Show();
```

Unlike interface members, dispinterface members cannot use the **retval** attribute to return a value in addition to an HRESULT error code. The **lcid** attribute is also invalid for dispinterfaces because **IDispatch::Invoke** passes a locale ID (LCID). However, it is possible to declare an interface again that uses these attributes.

Using Syntax 2, interfaces that support **IDispatch** and are declared earlier in an Object Definition Language (ODL) script can be redeclared as **IDispatch** interfaces as follows:

```
dispinterface helloPro {
    interface hello;
};
```

This example declares all of the members of the Hello sample and all of the members that it inherits to support **IDispatch**. In this case, if Hello was declared earlier with **lcid** and **retval** members that returned HRESULTs, MktypLib would remove each **lcid** parameter and HRESULT return type, and instead mark the return type as that of the **retval** parameter.

The properties and methods of a dispinterface are not part of the VTBL of the dispinterface. Consequently, **CreateStdDispatch** and **Displnvoke** cannot be used to implement **IDispatch::Invoke**. The dispinterface is used when an application needs to expose existing non-VTBL functions through OLE Automation. These applications can implement **IDispatch::Invoke** by examining the *dispidMember* parameter and directly calling the corresponding function.

Example

```
[uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0),
helpstring("Useful help string."), helpcontext(2480)]
dispinterface MyDispatchObject {
    properties:
        [id(1)] int x; // An integer property named x.
        [id(2)] BSTR y; // A string property named y.
```



```

    methods:
        [id(3)] void show();          // No arguments, no result.
        [id(11)] int computeit(int inarg, double *outarg);
};

[uuid 00000000-0000-0000-0000-123456789012]
dispinterface MyObject
{
    properties:
    methods:
        [id(1), propget, bindable, defaultbind, displaybind]
        long x();

        [id(1), propput, bindable, defaultbind, displaybind]
        void x(long rhs);
}

```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


enum Statement

Defines a C-style enumerated type.

Syntax

```
typedef [attributes] enum [tag] {  
    enumlist  
} enumname;
```

Syntax Elements

attributes

The **helpstring**, **helpcontext**, **hidden**, and **uuid** attributes are accepted before an **enum** statement. The **helpstring** and **helpcontext** attributes are accepted on an enumeration element. For more information about the attributes accepted before an enumeration definition, see “Attribute Descriptions” earlier in this chapter. Attributes (including the brackets) can be omitted. If **uuid** is omitted, the enumeration is not uniquely specified in the system.

tag

An optional tag, as with a C **enum**.

enumlist

List of enumerated elements.

enumname

Name by which the enumeration is known in the type library.

Comments

The **enum** keyword must be preceded by **typedef**. The enumeration description must precede other references to the enumeration in the library. If **value** is not specified for enumerators, the numbering progresses, as with enumerations in C. The type of the **enum** element is **int**, the system default integer, which depends on the target type library specification.

Examples

```
typedef [uuid(DEADF00D-CODE-B1FF-F001-A100FF001ED),  
        helpstring("Farm Animals are friendly"), helpcontext(234)]  
enum {  
    [helpstring("Moo")] cows = 1,  
    pigs = 2  
} ANIMALS;
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


importlib Directive

Makes types that have already been compiled into another type library available to the library currently being created. All **importlib** directives must precede the other type descriptions in the library.

Syntax

importlib(*filename*);

Syntax Element

filename

The location of the type library file when MkTypLib is executed.

Comments

The **importlib** directive makes any type defined in the imported library accessible from within the library being compiled. Ambiguity is resolved as the current library is searched for the type. If the type cannot be found, MkTypLib searches the imported library that is lexically first, and then the next, and so on. To import a type name in code, the name should be entered as *libname.type*, where *libname* is the library name as it appeared in the **library** statement when the library was compiled.

The imported type library should be distributed with the library being compiled.

Example

The following example imports the libraries Stdole.tlb and Mydisp.tlb:

```
library BrowseHelper
{
    importlib("stdole.tlb");
    importlib("mydisp.tlb");
    // Additional text omitted.
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

interface Statement

Defines an interface, which is a set of function definitions. An interface can inherit from any base interface.

Syntax

```
[attributes]
    interface interfacename [:baseinterface] {
        functionlist
    };
```

Syntax Elements

attributes

The attributes **dual**, **helpstring**, **helpcontext**, **hidden**, **odl**, **oleautomation**, **uuid**, and **version** are accepted before **interface**. If the interface is a member of a coclass, the attributes **source**, **default**, and **restricted** are also accepted. For more information about the attributes that can be accepted before an **interface** definition, refer to the section “Attribute Descriptions” earlier in this chapter.

The attributes **odl** and **uuid** are required on all **interface** declarations.

interfacename

The name by which the interface is known in the type library.

baseinterface

The name of the interface that is the base class for this interface.

functionlist

List of function prototypes for each function in the interface. Any number of function definitions can appear in the function list. A function in the function list has the following form:

```
[attributes] returntype [calling convention] funcname(params);
```

The following attributes are accepted on a function in an interface: **helpstring**, **helpcontext**, **string**, **propget**, **propput**, **propputref**, **bindable**, **defaultbind**, **displaybind**, and **vararg**. If **vararg** is specified, the last parameter must be a safe array of VARIANT type. The optional calling convention can be **__pascal/_pascal/pascal**, **__cdecl/_cdecl/cdecl**, or **__stdcall/_stdcall/stdcall**. The calling convention specification can include up to two leading underscores.

The parameter list is a comma-delimited list, as follows:

```
[attributes] type paramname
```

The *type* can be any previously declared type, built-in type, a pointer to any type, or a pointer to a built-in type. Attributes on parameters are **in**, **out**, **optional**, and **string**.

If **optional** is used, it must be specified only on the right-most parameters, and the types of those parameters must be VARIANT.

Comments

Because the functions described by the **interface** statement are in the VTBL, **DispInvoke** and **CreateStdDispatch** can be used to provide an implementation of **IDispatch::Invoke**. For this reason, **interface** is more commonly used than **dispinterface** to describe the properties and methods of an object.

Functions in interfaces are the same as described in “The module Statement,” except that the **entry** attribute is not allowed.

Members of interfaces that need to raise exceptions should return an HRESULT and specify a **retval** parameter for the actual return value. The **retval** parameter is always the last parameter in the list.

Examples

The following example defines an interface named Hello with two member functions, **HelloProc** and **Shutdown**:

```
[uuid(BFB73347-822A-1068-8849-00DD011087E8), version(1.0)]
interface hello : IUnknown
{
    void HelloProc([in, string] unsigned char * pszString);
    void Shutdown(void);
};
```

The next example defines a dual interface named **IMyInt**, which has a pair of accessor functions for the **MyMessage** property, and a method that returns a string.

```
[dual]
interface IMyInt : IDispatch
{
    // A property that is a string.
    [propget] HRESULT MyMessage([in, lcid] LCID lcid,
                                [out, retval] BSTR *pbstrRetVal);
    [propput] HRESULT MyMessage([in] BSTR rhs, [in, lcid] DWORD lcid);

    // A method returning a string.
    HRESULT SayMessage([in] long NumTimes,
                       [in, lcid] DWORD lcid,
                       [out, retval] BSTR *pbstrRetVal);
}
```

The members of this interface return error information and function return values through the HRESULT values and **retval** parameters, respectively. Tools that access the members can return the HRESULT to their users, or can simply expose the **retval** parameter as the return value, and handle the HRESULT transparently.

A dual interface must derive from **IDispatch**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


library Statement

Describes a type library. This description contains all of the information in a MkTypLib input file.

Syntax

```
[attributes] library libname {  
    definitions  
};
```

Syntax Elements

attributes

The **helpstring**, **helpcontext**, **lcid**, **restricted**, **hidden**, **control**, and **uuid** attributes are accepted before a **library** statement. For more information about the attributes accepted before a **library** definition, see "Attribute Descriptions" earlier in this chapter. The **uuid** attribute is required.

libname

The name by which the type library is known.

definitions

Descriptions of any imported libraries, data types, modules, interfaces, dispinterfaces, and coclasses relevant to the object being exposed.

Comments

The **library** statement must precede any other type definitions.

Example

```
[  
    uuid(F37C8060-4AD5-101B-B826-00DD01103DE1), // LIBID_Hello.  
    helpstring("Hello 2.0 Type Library"),  
    lcid(0x0409),  
    version(2.0)  
]  
library Hello  
{  
    importlib("stdole.tlb");  
    [  
        uuid(F37C8062-4AD5-101B-B826-00DD01103DE1), // IID_Ihello.  
        helpstring("Application object for the Hello application."),  
        oleautomation,  
        dual  
    ]  
    interface IHello : IDispatch  
    {  
        [propget, helpstring("Returns the application of the object.")]  
        HRESULT Application([in, lcid] long localeID,  
            [out, retval] IHello** retval)  
    }  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


module Statement

Defines a group of functions, typically a set of DLL entry points.

Syntax

```
[attributes]
    module modulename {
        elementlist
    };
```

Syntax Elements

attributes

The attributes **uuid**, **version**, **helpstring**, **helpcontext**, **hidden**, and **dllname** are accepted before a **module** statement. For more information about the attributes that can be accepted before a module definition, see “Attribute Descriptions” earlier in this chapter. The **dllname** attribute is required. If **uuid** is omitted, the module is not uniquely specified in the system.

modulename

The name of the module.

elementlist

List of constant definitions and function prototypes for each function in the DLL. Any number of function definitions can appear in the function list. A function in the function list has the following form:

```
[attributes] returntype [calling convention] funcname(params);
[attributes] const constname = constval;
```

Only the attributes **helpstring** and **helpcontext** are accepted for a **const**.

The following attributes are accepted on a function in a module: **helpstring**, **helpcontext**, **string**, **entry**, **propget**, **propput**, **propputref**, **vararg**. If **vararg** is specified, the last parameter must be a safe array of VARIANT type.

The optional *calling convention* can be one of **__pascal/_pascal/pascal**, **__cdecl/_cdecl/cdecl**, or **__stdcall/_stdcall/stdcall**. The calling convention can include up to two leading underscores.

The parameter list is a comma-delimited list.

[attributes] type paramname

The *type* can be any previously declared type or built-in type, a pointer to any type, or a pointer to a built-in type. Attributes on parameters are **in**, **out**, and **optional**.

If **optional** is specified, it must only be specified on the right-most parameters, and the types of those parameters must be VARIANT.

Comments

The header file (.h) output for modules is a series of function prototypes. The **module** keyword and surrounding brackets are stripped from the header file output, but a comment (`\\ module modulename`) is inserted before the prototypes. The keyword **extern** is inserted before the declarations.

Example

```
[uuid(D00BED00-CEDE-B1FF-F001-A100FF001ED),
```



```
    helpstring("This is not GDI.EXE"), helpcontext(190),  
    dllname("MATH.DLL")]  
module somemodule{  
    [helpstring("Color for the frame")] unsigned long const COLOR_FRAME  
        = 0xH800000006;  
    [helpstring("Not a rectangle but a square"), entry(1)] pascal double  
    square([in] double x);  
};
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


struct Statement

Defines a C-style structure.

Syntax

```
typedef [attributes]
    struct [tag] {
        memberlist
    } structname;
```

Syntax Elements

attributes

The attributes **helpstring**, **helpcontext**, **uuid**, **hidden**, and **version** are accepted before a **struct** statement. The attributes **helpstring**, **helpcontext**, and **string** are accepted on a structure member. For more information about the attributes accepted before a structure definition, see “Attribute Descriptions” earlier in this chapter. Attributes (including the brackets) can be omitted. If **uuid** is omitted, the structure is not specified uniquely in the system.

tag

An optional tag, as with a C **struct**.

memberlist

List of structure members defined with C syntax.

structname

Name by which the structure is known in the type library.

Comments

The **struct** keyword must be preceded with a **typedef**. The structure description must precede other references to the structure in the library. Members of a **struct** can be of any built-in type, or any type defined lexically as a **typedef** before the **struct**. For a description of how strings and arrays can be entered, see the sections “String Definitions” and “Array Definitions” earlier in this chapter.

Example

```
typedef [uuid(BFB7334B-822A-1068-8849-00DD011087E8),
    helpstring("A task"), helpcontext(1019)]
struct {
    DATE startdate;
    DATE enddate;
    BSTR ownername;
    SAFEARRAY (int) subtasks;
    int A_C_array[10];
} TASKS;
```

{**ewl** msdncd.dll, **ewcright**, /c"Microsoft"}

typedef Statement

Creates an alias for a type.

Syntax

typedef [attributes] basetype aliasname;

Syntax Elements

attributes

Any attribute specifications must follow the **typedef** keyword. If no attributes and no other type (for example, **enum**, **struct**, or **union**) are specified, the alias is treated as a **#define** and does not appear in the type library. If no other attribute is desired, **public** can be used to explicitly include the alias in the type library. The **helpstring**, **helpcontext**, and **uuid** attributes are accepted before a **typedef**. For more information, see “Attribute Descriptions” earlier in this chapter. If **uuid** is omitted, the **typedef** is not uniquely specified in the system.

basetype

The type for which the alias is defined.

aliasname

Name by which the type will be known in the type library.

Comments

The **typedef** keyword must also be used whenever a **struct** or **enum** is defined. The name recorded for the **enum** or **struct** is the **typedef** name, and not the tag for the enumeration. No attributes are required to make sure the alias appears in the type library.

Enumerations, structures, and unions must be defined with the **typedef** keyword. The attributes for a type defined with **typedef** are enclosed in brackets following the **typedef** keyword. If a simple alias **typedef** has no attributes, it is treated like a **#define**, and the *aliasname* does not appear in the library. Any attribute (**public** can be used if no others are desired) specified between the **typedef** keyword and the rest of a simple alias definition causes the alias to appear explicitly in the type library. The attributes typically include such items as a Help string and Help context.

Examples

```
typedef [public] long DWORD;
```

This example creates a type description for an alias type with the name **DWORD**.

```
typedef enum {  
    TYPE_FUNCTION = 0,  
    TYPE_PROPERTY = 1,  
    TYPE_CONSTANT = 2,  
    TYPE_PARAMETER = 3  
} OBJTYPE;
```

The second example creates a type description for an enumeration named **OBJTYPE**, which has four enumerator values.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


union Statement

Defines a C-style union.

Syntax

```
typedef [attributes] union [tag] {  
    memberlist  
} unionname;
```

Syntax Elements

attributes

The attributes **helpstring**, **helpcontext**, **uuid**, **hidden**, and **version** are accepted before a **union**. The **helpstring**, **helpcontext**, and **string** attributes are accepted on a **union** member. For more information about the attributes accepted before a union definition, see “Attribute Descriptions” earlier in this chapter. Attributes (including the square brackets) can be omitted. If **uuid** is omitted, the union is not uniquely specified in the system.

tag

An optional tag, as with a C **union**.

memberlist

List of **union** members defined with C syntax.

unionname

Name by which the **union** is known in the type library.

Comments

The **union** keyword must be preceded with a **typedef**. The **union** description must precede other references to the structure in the library. Members of a **union** can be of any built-in type, or any type defined lexically as a **typedef** before the **union**. For a description of how strings and arrays can be entered, see the sections “String Definitions” and “Array Definitions” earlier in this chapter.

Example

```
[uuid(BFB7334C-822A-1068-8849-00DD011087E8), helpstring("A task"),  
helpcontext(1019)]  
typedef union {  
    COLOR polycolor;  
    int    cVertices;  
    boolean filled;  
    SAFEARRAY (int) subtasks;  
} UNIONSHOP;
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Copyright List

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

© 1996 Microsoft Corporation. All rights reserved.

Microsoft® Visual J++™

Development System for Windows® 95 and Windows NT™

Microsoft Corporation

Microsoft, MS, MS-DOS, FoxPro, Microsoft Press, Visual Basic, Windows, Win32, and Win32s are registered trademarks and DataTips, Visual J++, and Windows NT are trademarks of Microsoft Corporation in the U.S. and other countries.

{ewc msdn cd, EWGraphic, cop0a 0 /a "copy.BMP"}

Java is a registered trademark of Sun Microsystems, Inc.

Apple, Appletalk, Macintosh, MacOS, MPW, and TrueType are registered trademarks and Macintosh Quadra, QuickDraw, and Power Macintosh are trademarks of Apple Computer, Inc.

Borland, dBASE, dBASE II, dBASE III, and dBASE IV are registered trademarks, and Paradox is a trademark of Borland International, Inc.

BRIEF is a registered trademark of SDC Software Partners II L.P.

Btrieve, NetWare, and Novell are registered trademarks of Novell, Inc.

CompuServe is a registered trademark of CompuServe, Inc.

DEC and Alpha AXP are trademarks of Digital Equipment Corporation.

Epsilon is a trademark of Lugaru Software, Inc.

Finder is a trademark of Apple Computer, Inc.

GENie is a trademark of General Electric Corporation.

Hewlett-Packard and LaserJet are registered trademarks of Hewlett-Packard Company.

HyperCard is a registered trademark of Claris Corporation.

IBM and OS/2 are registered trademarks and PowerPC, PowerPC 601, and PowerPC 603 are trademarks of International Business Machines Corporation.

Intel and Pentium are registered trademarks and i486 is a trademark of Intel Corporation.

MIPS is a registered trademark of MIPS Computer Systems, Inc.

Motorola is a registered trademark of Motorola, Inc.

OpenGL is a trademark of Silicon Graphics, Inc.

ORACLE is a registered trademark of Oracle Corporation.

Paintbrush is a trademark of Wordstar Atlanta Technology Center.

SYBASE is a registered trademark of Sybase, Inc.

Unicode is a trademark of Unicode, Incorporated.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0001

INTERNAL COMPILER ERROR: '*identifier*'

The compiler was unable to recover after detecting an error. If you receive this message, consult your technical support help file for information on how to address this problem.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0002

Out of memory

The compiler attempted to allocate some additional memory during processing, but was unable to do so. When this error occurs, check the location, size and validity of your system swap file. Also, check that there is sufficient growth space available on the drive on which your swap file resides.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0003

Invalid code: '*string*'

The compiler may have detected generation of invalid code. Try dividing larger methods defined within your classes into smaller methods and compiling again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0004

Cannot open class file '*filename*' for reading

The compiler could not open the program source file for reading. This error most likely occurs when another program has an exclusive lock on the source file. Try shutting down other processes that may be accessing the source file and compiling again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0005

Cannot open class file '*filename*' for writing

The compiler failed to generate the output .CLASS file. This error most likely occurs when the compiler cannot get write or create permission for the file. Make sure the file does not have its read-only attribute set, and that it is not currently in use by another a process.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0006

Cannot read class file '*filename*'

The compiler failed to read the specified .CLASS file. This error most likely occurs when the compiler encounters an error reading the storage device, or when the compiler cannot otherwise get read permission for the file. Check to ensure the file is not currently in use by another process. Also, use whatever means available (i.e. SCANDISK) to ensure the validity of the storage device you are attempting to use.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0007

Cannot write class file '*filename*'

The compiler failed while attempting to write the contents of a buffer to the specified .CLASS file.

This error most likely occurs when space is exhausted on the targeted storage device. Try freeing up any available space on the storage device and compiling again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0008

Cannot locate class file '*filename*'

The compiler could not locate a core language class for the **java.lang** package. This error most likely occurs when the Java **CLASSPATH** variable is not properly set . Try correcting the CLASSPATH variable, and compiling again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0010

Syntax error

The compiler could not determine the meaning of an expression or statement within the source program. This error most likely occurs when the line indicated in the error message is syntactically invalid. This error usually accompanies a more descriptive error. Try correcting any accompanying errors and compiling again.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i =; // error: initialization error  
    }  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0011

Expected ':'

The compiler expected to find a colon following a **case** label or in a conditional expression that makes use of the ternary operator. This error most likely occurs when the colon is accidentally omitted.

The following sample illustrates this error:

```
public class Simple {  
  
    private int i;  
    private static int x = 1;  
  
    public void method1(int arg1) {  
  
        switch (arg1) {  
            case 1 // error: ':' omitted  
                ; // do something meaningful  
        }  
  
        i = ( arg1 < x) ? arg1  x; // error: ':' omitted  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0012

Expected ';'

The compiler expected to find a semicolon in the position indicated by the error message. This error most likely occurs when the semicolon is accidentally omitted from the end of a statement. This error can also occur when a conditional expression is not syntactically correct.

The following sample illustrates this error:

```
public class Simple {  
    private static int x = 10 // error: ';' omitted  
  
    public void method1(int arg1) {  
        for (int i = 1; i < x i++) {  
            // error: ';' omitted  
            ;  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0013

Expected '('

The compiler expected to find a left parenthesis in the position indicated by the error message. This error most likely occurs when the left parenthesis is accidentally omitted in any of the following situations:

- type initializations
- **catch** statements
- parenthesized expressions
- **while** loops
- **for** loops
- **if-else** statements

The following sample illustrates this error:

```
public class Simple {  
    private int i;  
    public void method1(int arg1, int arg2) {  
        if arg1 < arg2) // error: '(' omitted  
            i = arg1;  
        else  
            i = arg2;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0014

Expected ')'

The compiler expected to find a right parenthesis in the position indicated by the error message. This error most likely occurs when the right parenthesis is accidentally omitted in any of the following situations:

- type initializations
- type casts
- **catch** statements
- parenthesized expressions
- **while** loops
- **for** loops
- **if-else** statements

The following sample illustrates this error:

```
public class Simple {  
  
    private int i;  
  
    public void method1(int arg1, int arg2) {  
  
        i = (arg1 < arg2 ? arg1 : arg2;  
        // error: ')' omitted  
    }  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0015

Expected ']'

The compiler expected to find a right square bracket in the position indicated by the error message. This error most likely occurs when the right square bracket is accidentally omitted from an array declaration.

The following sample illustrates this error:

```
public class Simple {  
    private int x[ = new int[500]; // error: ']' omitted  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0016

Expected '{'

The compiler expected to find a left brace in the position indicated by the error message. This error most likely occurs when the left brace is accidentally omitted from the beginning of a class declaration.

The following sample illustrates this error:

```
public class Simple // error: '{' omitted

    public void method1() {
        // do something meaningful
    }
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0017

Expected '}'

The compiler expected to find a right brace in the position indicated by the error message. This error most likely occurs when a statement containing a right brace is not syntactically correct.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int arr[] = {1, 2A};  
        // error: invalid initialization  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0018

Expected 'while'

The compiler expected to find the keyword **while** in the position indicated by the error message. This error most likely occurs when a **do/while** loop is not syntactically correct. The correct structure for a **do/while** loop is:

```
do {  
    // do something useful here  
} while (condition);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0019

Expected identifier

The compiler expected to find an identifier toward the end of a class, interface, variable, or method declaration. This error most likely occurs when the type is accidentally omitted in a declaration.

The following sample illustrates this error:

```
public class Simple {  
    private i; // error: type omitted  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0020

Expected 'class' or 'interface'

The compiler expected to find either **class** or **interface** used within the corresponding declaration. This error most likely occurs when the keywords are accidentally omitted from a **class** or **interface** declaration. Another possible cause of this error is unbalanced scoping braces.

The following sample illustrates this error:

```
public class Simple {  
    // do something meaningful  
}}    // error: additional '}' is the problem
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0021

Expected type specifier

The compiler expected to find a type specifier in the position indicated by the error message.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0022

Expected end of file

The compiler expected to encounter an end of file character, but did not. This error most likely occurs when the source file has been damaged in some way. Try visually checking the source file for obvious corruption, save any changes and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0023

Expected 'catch' or 'finally'

The compiler expected to find a **catch** or **finally** block immediately following a corresponding **try** block.

The following sample illustrates this error:

```
public class Simple {  
    public void method1( ) {  
        try {  
            // do something meaningful  
        }  
    } // error: 'catch' or 'finally' not found  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0024

Expected method body

The compiler expected to find a method body immediately following a method declaration. This error most likely occurs when the braces surrounding the method body are not properly balanced. This error may also occur when the method was intended to be **abstract**, but the **abstract** keyword was mistakenly omitted from the method declaration.

The following sample illustrates this error:

```
public abstract class Simple {  
  
    public void method1( );  
    // error: 'abstract' omitted  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0025

Expected statement

The compiler expected to find a statement before the end of the current scope. This error most likely occurs when the right brace designating the end of the current scope is misplaced.

The following sample illustrates this error:

```
abstract class Simple {  
    void method1() {  
        if (1)  
            // error: a statement is required  
            // after the if statement  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0026

Expected Unicode escape sequence

The compiler expected to find a valid Unicode escape sequence. This error most likely occurs when a syntactical error is found in a Unicode escape sequence.

The following sample illustrates this error:

```
public class Simple {  
    int i = \\u0032;  
    // error: '\\\' not invalid  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0027

Identifier too long

The compiler detected an identifier name with a length greater than 1024 characters. Shorten the identifier name and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0028

Invalid number

The compiler detected a numeric value that the Java language is not capable of supporting. This error most likely occurs when the number specified is an amount greater than any of Java's primitive types can accept.

The following sample illustrates this error:

```
public class Simple {  
    long i = 12345678901234567890;  
    // error: value out of range  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0029

Invalid character

The compiler detected an ASCII character that could not be used in an identifier. This error most likely occurs when a class, interface, method, or variable identifier includes an invalid character.

The following sample illustrates this error:

```
public class Simple {  
    private int c#;  
    // error: '#' not supported  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0030

Invalid character constant

The compiler detected an attempt to assign an invalid character or character escape sequence to a variable of type **char**.

The following sample illustrates this error:

```
public class Simple {  
    char c = '\\';  
    // error: invalid escape character  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0031

Invalid escape character

The compiler detected the use of an invalid escape character. This error most likely occurs when a syntactical error is found in a Unicode escape sequence.

The following sample illustrates this error:

```
public class Simple {  
  
    int i = \u032;  
    // error: Unicode uses 4 hex digits  
  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0032

Unterminated string constant

The compiler did not detect a terminating double-quote character at the end of a string constant. This error most likely occurs when the string terminator is accidentally omitted, or when the string constant is mistakenly divided onto multiple lines.

The following sample illustrates this error:

```
public class Simple {  
    String str = "Hello  
    // error: '"' and ';' omitted  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0033

Unterminated comment

The compiler detected the beginning of a block comment, but did not detect a valid ending for it. This error most likely occurs when the comment terminator is accidentally omitted.

The following sample illustrates this error:

```
public class Simple {  
    /* This comment block  
    * does not have a valid  
    * terminator  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0035

Initializer block must be declared 'static'

The compiler detected a possible static initializer block, but did not detect the keyword **static** immediately preceding it. This error most likely occurs when the keyword **static** is accidentally omitted. This error may also occur when some other syntactical error exists.

The following sample illustrates this error:

```
public class Simple {  
    static private int i;  
  
    {    // error: 'static' omitted  
        i = 1;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0036

A data member cannot be 'native', 'abstract' or 'synchronized'

The compiler detected one of the modifiers shown above used in the declaration of a variable. The modifiers **synchronized** and **native** can only be applied to method declarations. The **abstract** modifier can be applied to methods, classes, and interfaces.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0037

A method cannot be 'transient' or 'volatile'

The compiler detected one of the modifiers shown above used in the declaration of a method. The modifiers **transient** and **volatile** can only be applied to variable declarations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0038

'final' members must be initialized

The compiler detected an uninitialized **final** variable. Variables declared as **final** must have their value set at declaration. Once set, the value cannot be programmatically changed.

The following sample illustrates this error:

```
public class Simple {  
  
    private final int MAX_HEADROOM;  
    // error: must have value set  
  
}
```

Note that variables declared within interfaces are implicitly defined as **final** or **static**. As such, this error also occurs when their initial values are not set at declaration.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0040

Cannot define body for abstract/native methods

The compiler detected a method body defined immediately following the corresponding declaration of an **abstract** or **native** method.

The following sample illustrates this error:

```
public interface Simple {  
  
    public void method1() {  
        // error: must define this method body  
        // in a class that implements the  
        // 'Simple' interface  
    }  
}
```

Note that methods declared within an interface are implicitly **abstract**. As such, this error will also occur when you attempt to define their body in an interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0041

Duplicate modifier

The compiler detected a modifier used twice in a declaration. This error most likely occurs when the same modifier is mistakenly used more than once within a declaration.

The following sample illustrates this error:

```
public class Simple {  
    public public void method1() { // error: 'public' used twice  
        // do something meaningful  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0042

Only classes can implement interfaces

The compiler detected an interface declaration using the **implements** keyword. Interfaces cannot implement other interfaces. Rather, interfaces may only be implemented by classes.

The following sample illustrates this error:

```
public interface Simple implements color{
    // error: 'Simple' cannot implement the
    // 'color' interface
}
interface color {
    // do something meaningful
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0043

Redeclaration of member '*identifier*'

The compiler detected the same identifier name being declared more than once within the same scope. This error most likely occurs when a variable is mistakenly declared more than once.

The following sample illustrates this error:

```
public class Simple {  
    private int i;  
    private int i; // error: 'i' declared twice  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0044

Cannot find definition for class '*identifier*'

The compiler could not locate the definition for the specified class. This error is most likely caused by a typographical error. It may also occur when the package containing the specified class cannot be found.

The following sample illustrates this error:

```
public class Simple {  
    char buf[] = new chaar[5];  
    // error: 'chaar' not a valid type  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0045

'identifier' is not a class name

The compiler detected one of the following conditions:

- A class name provided as part of an **import** statement could not be found or the import statement was not syntactically valid.
- The class attempted to extend an interface.

The following sample illustrates this error:

```
package non.existent;  
  
import non.existent; // error:  
  
public class Simple {  
    // do something meaningful  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0046

'identifier' is not an interface name

The compiler detected that the identifier referred to by the keyword **implements** is not an interface.

The following sample illustrates this error:

```
class Simple2 {  
    // do something meaningful  
}  
  
public class Simple implements Simple2 {  
    // error: cannot implement class 'Simple2'  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0047

'*identifier*' is not a package name

The compiler detected an invalid package name. This error most likely occurs when a syntactical error exists in an **import** statement, or when the package name does not otherwise exist.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0048

Cannot extend final class '*identifier*'

The compiler detected an attempt to subclass a class declared with the keyword **final**. Classes declared as **final** cannot be subclassed.

The following sample illustrates this error:

```
final class Simple2 {  
    // do something meaningful  
}  
  
public class Simple extends Simple2 {  
    // error: cannot extend 'Simple2'  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0049

Undefined name '*identifer*'

The compiler detected an unknown class name while processing an **import** statement. This error most likely occurs when the identifier is misspelled or does not exist. This error may also occur if the **CLASSPATH** variable is not set correctly.

The following sample illustrates this error:

```
import java.io.bogus; // error: unknown class name

public class Simple {

    // do something meaningful

}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0050

'*identifier*' is not a member of '*identifier*'

The compiler detected a reference to an identifier that is not a member of the specified package. This error most likely occurs when the identifier is misspelled or does not exist.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        java.lang.bogus.method1();  
        // error: 'bogus' not member of 'lang'  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0051

Undefined package '*identifier*'

The compiler detected a package name, but was unable to locate the package definition. This error most likely occurs when a syntactical error exists in an **import** statement. This error may also occur when the package cannot be found.

The following sample illustrates this error:

```
import java.lang.String.*;
// error: 'String' not a valid package name

public class Simple {

    // do something meaningful

}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0053

Ambiguous name: '*identifier*' and '*identifier*'

The compiler could not resolve an ambiguity between the two identifiers shown. Correct the ambiguity between the two types and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0054

Methods '*identifier*' and '*identifier*' differ only in return type

The compiler detected two or more method overloads having identical parameters. In Java, overloaded methods must be distinguished by unique signatures. A unique signature consists of the method name, the number, type and positions of its parameters, and the return type.

The following sample illustrates this error:

```
public class Simple {  
    public int method1(int arg1, char arg2) {  
        // do something meaningful  
        return arg1;  
    }  
  
    public char method1(int arg3, char arg4) {  
        // error: identical parameters  
        return arg4;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0055

A constructor may not specify a return type

The compiler detected a constructor declaration specifying a return type. Constructors must implicitly return an instance of the declared class. As such, their return type may not be specified.

The following sample illustrates this error:

```
public class Simple {  
  
    int Simple() {  
        // error: return type specified  
        return 1;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0056

Missing return type specification

The compiler detected a method declaration without a return type specified. All method declarations must specify a return type. If the method is not meant to return a value, use the **void** keyword.

The following sample illustrates this error:

```
public class Simple {  
    public method1() { // error: no return type  
        // do something meaningful  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0057

Class file '*identifier*' doesn't contain class '*identifier*'

The compiler did not detect the class name shown above within the specified file. This error most likely occurs when the class name is either misspelled or does not exist. This error may also occur when a .CLASS file has been renamed after successful compilation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0058

Cannot have a variable of type 'void'

The compiler detected a variable declared as type void. The keyword **void** is not allowed in variable declarations. Rather, void can only be used as a method return type, noting that the method does not actually return a value.

The following sample illustrates this error:

```
public interface Simple {  
    public final static void i = 1;  
    // error: 'void' not valid  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0059

Cannot reference member '*identifier*' without an object

The compiler detected an attempt to reference a variable without a known object association. This error most likely occurs when an instance variable (a variable declared without the keyword **static**) is referenced from within a class (or static) method.

The following sample illustrates this error:

```
public class Simple {  
    private int x;  
  
    public static void method1() {  
        x = 0; // error: 'x' must be static  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0060

Invalid forward reference to member '*identifier*'

The compiler detected an attempt to initialize a variable with another variable that had not yet been defined.

The following sample illustrates this error:

```
public class Simple {  
    private static int i = j;  
    // error: 'j' not yet defined  
    private static int j = 0;  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0061

The members '*identifier*' and '*identifier*' differ in return type only

The compiler detected a subclass method attempting to overload a base class method, but the methods differed only in return type. In Java, overloaded methods must be distinguished by unique signatures. A unique signature consists of the method name, the number, type and positions of its parameters, and the return type.

The following sample illustrates this error:

```
public class Simple extends Simple2 {  
    public void method1() {  
        // error: only return type differs  
    }  
}  
  
class Simple2 {  
    public int method1() {  
        return 1;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0062

Attempt to reduce the access level of member '*identifier*'

The compiler detected an overloading method in the class being compiled that reduces the access level of its base class method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0063

Declare the class abstract, or implement abstract member '*identifier*'

The compiler detected the declaration of an abstract method within a class, but the class was not declared to be abstract.

The following sample illustrates this error:

```
public class Simple {  
  
    public abstract void method1();  
    // error: class not abstract  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0064

Local variable '*identifier*' shadows another local variable

The compiler detected two or more variables with the same identifier defined within the same scope of a method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0065

Cannot assign to this expression

The compiler detected an expression in the position normally held by an lvalue for assignment.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int x = 0;  
        int y = 1;  
        x++ = y; //error: '++' not valid  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0066

'this' can only be used in non-static methods

The compiler detected use of the keyword **this** within a class (or static) method. Class methods are not passed implicit **this** references. As such, they cannot reference instance (non-static) variables or methods.

The following sample illustrates this error:

```
public class Simple {  
    int x;  
  
    public static void method1() {  
        this.x = 12; // error: 'this' instance specific  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0067

Cannot convert '*identifier*' to '*identifier*'

The compiler detected a variable type used out of its correct context. As such, the compiler could not implicitly convert the result to anything meaningful.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i = 10;  
        if (i--) {  
            // error: conditional needs expression  
            // or boolean variable  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0068

Cannot implicitly convert '*identifier*' to '*identifier*'

The compiler could not convert the specified variable without an explicit type cast.

The following sample illustrates this error:

```
public class Simple {  
    int i = 10;  
  
    public char method1() {  
        return i; // error: expected type char  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0069

Cannot apply '.' operator to an operand of type '*identifier*'

The compiler detected the '.' operator applied to an invalid type. This error most likely occurs when the .length method is applied to an invalid type.

The following sample illustrates this error:

```
public class Simple {  
    int i = 123;  
    int j = i.length; // error: 'i' not an array  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0070

Need argument list for call to member '*identifier*'

The compiler detected syntax for a known method call, but did not detect an associated argument list. This error most likely occurs when a syntactical error exists in the call.

The following sample illustrates this error:

```
public class Simple {  
    public static void main(String args[]) {  
        System.out.println = ("Hello");  
        // error: '=' invalid  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0071

Cannot use argument list with '*identifier*'

The compiler detected syntax for a method call, but the identifier found could not be a method. This error most likely occurs when a method name is misspelled and matches a variable name.

The following sample illustrates this error:

```
class Simple {  
    int x;  
    int y = x();  
    //error: x is not a method  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0072

'*identifier*' is not a member of class '*identifier*'

The compiler detected a method call, but the method remains undefined. This error most likely occurs when the method name is either misspelled, or cannot be found within proper scope. This error may also occur when the method does not exist.

The following sample illustrates this error:

```
public class Simple {  
    public static void main(String args[]) {  
        System.out.println("Hello");  
        // error: 'println' should be 'println'  
    }  
}
```

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Compiler Error J0073

'*identifier*' cannot access member '*identifier*'

The compiler detected an invalid attempt to access a class member within a different package.
Correct the visibility of the member being accessed and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0074

Operator cannot be applied to '*identifier*' and '*identifier*' values

The compiler detected an operator that cannot be correctly applied to the identifiers shown in the error message.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        String s1 = "one";  
        String s2 = "two";  
        String result;  
        result = s1 * s2; // error: invalid operands  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0075

Invalid call

The compiler detected syntax for a method call, but the identifier does not represent a valid method name. This error most likely occurs when a method name is misspelled or contains characters that are not recognized as valid in Java naming conventions.

The following sample illustrates this error:

```
class Simple {  
    int x = 1();  
    //error: 1 is not a valid method name  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0076

Too many arguments for method '*identifier*'

The compiler detected a known method call, but the call contained more arguments than needed.

The following sample illustrates this error:

```
public class Simple {  
    public void method1(int arg1) {  
        // do something meaningful  
    }  
  
    public void method2() {  
        method1(1, 2); // error: Too many arguments  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0077

Not enough arguments for method '*identifier*'

The compiler detected a known method call, but the call contained fewer arguments than needed. This error most likely occurs when one or more arguments are accidentally omitted from the call.

The following sample illustrates this error:

```
public class Simple {  
  
    public void method1(int arg1) {  
        // do something meaningful  
    }  
  
    public void method2() {  
        method1(); // error: Too few arguments  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0078

Class '*identifier*' doesn't have a method that matches '*identifier*'

The compiler identified a call to a known overloaded method within another class, but was unable to detect a matching method with the correct number of arguments.

The following sample illustrates this error:

```
public class Simple {  
  
    public void method1() {  
        // do something meaningful  
    }  
  
    public void method1(int arg1) {  
        // do something meaningful  
    }  
}  
  
class Simple2 {  
  
    public void method1() {  
  
        Simple s = new Simple();  
        s.method1(1, 2, 3); // error: too many args  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0079

Ambiguity between '*identifier*' and '*identifier*'

The compiler could not distinguish the correct method to execute. This error most likely occurs when the method parameters specified in a call do not exactly match any of those defined within the called methods declaration, or any of its base classes.

The following sample illustrates this error:

```
public class Simple {

    static void method1(Simple2 s2, Simple3 s3) {
        // do something meaningful
    }

    static void method1(Simple3 s3, Simple2 s2) {
        // do something meaningful
    }

    public static void main(String args[]) {

        Simple2 s2 = new Simple2();
        method1(s2, s2);
        // error: Ambiguity between Simple2 and Simple3
    }
}

class Simple2 extends Simple3 {
    // do something meaningful
}

class Simple3 {
    // do something meaningful
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0080

Value for argument '*identifier*' cannot be converted from '*identifier*' in call to '*identifier*'

The compiler detected a method argument that does not match the parameters specified in the method declaration.

The following sample illustrates this error:

```
public class Simple {  
  
    public void method1(int arg1) {  
        // do something meaningful  
    }  
  
    public void method2() {  
  
        float f = 1.0f;  
        method1(f); // error: mismatched call types  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0081

Value for argument '*identifier*' cannot be converted from '*identifier*' in call to '*identifier*'

The compiler detected a method call, but was unable to convert one of the arguments from the supplied type to the type shown in the method declaration. This error most likely occurs when a method is called with the arguments in the wrong order, or the wrong method was called.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0082

Class '*identifier*' doesn't have a constructor that matches '*identifier*'

The compiler did not detect a constructor matching the call identified in the error. This error most likely occurs when a constructor is called with the wrong number of arguments.

The following sample illustrates this error:

```
public class Simple {  
  
    Simple(int arg1) {  
        // do something meaningful  
    }  
  
    public static void main (String args[]) {  
  
        Simple s = new Simple(12, 13);  
        // error: too many arguments  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0083

'super()' may only be called within a constructor

The compiler detected use of the keyword **super** within a method. This keyword can only be used within a constructor.

The following sample illustrates this error:

```
public class Simple {  
    public void method1 () {  
        super(); // error: 'super' cannot be called  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0084

Can't return a value from a 'void' method

The compiler detected an attempt to return a value from a method declared with a return type of **void**.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        return 1; // error: cannot return a value  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0085

Expected return value of type '*identifier*'

The compiler detected the keyword **return** within the body of a method which was declared to return a specific type, but the **return** had no associated value.

The following sample illustrates this error:

```
public class Simple {  
    public int method1() {  
        return; // error: must return int value  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0086

'[]' cannot be applied to a value of type '*identifier*'

The compiler detected array brackets used with a non-array variable type.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i = 0;  
        int j, x;  
  
        x = j[i]; // error: 'j' not declared as array  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0087

The 'goto' statement is not currently supported by Java

The keyword **goto**, while defined as a keyword, has not yet been implemented in the Java language.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0089

Already had 'case: '*identifier*'

The compiler identified two or more **case** statements with the same identifier or value occurring within the same **switch** statement.

The following sample illustrates this error:

```
public class Simple {  
    public int method1(int arg1) {  
        switch (arg1) {  
            case 1:  
                return (int) 1;  
            case 2:  
                return (int) 2;  
            case 2: // error: duplicate of above  
                return (int) 3;  
            default:  
                return (int) 0;  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0090

Already had 'default'

The compiler identified two or more instances of the keyword **default** occurring within the same **switch** statement.

The following sample illustrates this error:

```
public class Simple {  
    public int method1(int arg1) {  
        switch (arg1) {  
            case 1:  
                return (int) 1;  
            case 2:  
                return (int) 2;  
            default:  
                return (int) 3;  
            default: // error: duplicate of above  
                return (int) 0;  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0091

'case' outside of switch statement

The compiler identified the keyword **case** used outside the scope of a **switch** statement.

The following sample illustrates this error:

```
public class Simple {  
    public int method1() {  
        case 1: // error: no switch statement  
            return 1;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0092

Constant expression expected

The keyword **const**, while defined as a keyword, has not yet been implemented in the Java language.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0093

'break' only allowed in loops and switch statements

The compiler detected the keyword **break** occurring outside the scope of a loop or **switch** statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0094

Label '*identifier*' not found

The compiler detected a label name associated with one of the keywords **continue** or **break**, but could not find the label.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0095

'continue' only allowed in loop

The compiler detected attempted use of the keyword **continue** outside the scope of a loop.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0096

Class value expected

The compiler detected a synchronization block, but the **synchronized** modifier was applied to an invalid type.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i;  
        synchronized (i);  
        // error: 'i' must resolve to  
        // class or array type  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0097

Class or array expected

The compiler detected the **instanceof** operator applied to a type that did not resolve to a class or array.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        Simple2 obj = new Simple2();  
        if (obj instanceof int) // error: bad type  
            ; // do something meaningful  
    }  
}  
  
class Simple2 {  
    // do something meaningful  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0098

Attempt to access non-existent member of '*identifier*'

The compiler detected an array member specified, but could not identify it.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int j[] = new int[10];  
        int i = j.bogus;  
        // error: 'bogus' not valid member  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0100

Cannot throw '*identifier*' - the type doesn't inherit from 'Throwable'

The compiler detected an object in a **throw** statement that was not derived from the class `Throwable`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0101

The type '*identifier*' does not inherit from 'Throwable'

The compiler detected an invalid class argument used as an argument in a **catch** declaration.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        try {  
            // do something meaningful  
        } catch (String s) {  
            // error: 'String' not a subclass  
            // of 'Throwable'  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0102

Handler for '*identifier*' hidden by earlier handler for '*identifier*'

The compiler detected an exception handler that will never be executed because an earlier handler would have already caught the exception. This error is most likely caused by putting **catch** statements in the wrong order.

The following sample illustrates this error:

```
class Simple {  
    static  
    {  
        try  
        {  
        }  
        catch (Exception e)  
        {  
        }  
        catch (ArithmeticException e)  
        {  
        }  
        // error: any exceptions this block  
        // could have caught are already caught  
        // by the first catch statement.  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0103

Cannot override final method '*identifier*'

The compiler detected a class method attempting to override one of its base class methods, but the base class method was declared with the keyword **final**.

The following sample illustrates this error:

```
public class Simple extends Simple2 {  
    public void method1() {  
        // error: 'method1' final in superclass  
    }  
}  
  
class Simple2 {  
    public final void method1() {  
        // do something meaningful  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0104

Unreachable statement or declaration

The compiler detected a statement or declaration that cannot be reached under any circumstances.

The following sample illustrates this error:

```
class Simple {  
    static  
    {  
        return;  
        int x;  
        //error: the declaration cannot be reached  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0105

Method '*identifier*' must return a value

The compiler detected a method declaration which included a return type other than **void**, but the keyword **return** was not found in the method body.

The following sample illustrates this error:

```
public class Simple {  
    public int method1(int arg1) {  
        } // error: expected an integer returned  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0106

Class '*identifier*' has a circular dependency

The compiler detected two or more classes directly or indirectly attempting to subclass each other.

The following sample illustrates this error:

```
public class Simple extends Simple2 {  
    // error: extending 'Simple2'  
}  
  
class Simple2 extends Simple {  
    // error: also extending 'Simple'  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0107

Missing array dimension

The compiler detected the initialization of an array, but failed to detect a valid array dimension.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int [][] i = new int[][12];  
        // error: missing first array dimension  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0108

Cannot 'new' an instance of type '*identifier*'

The compiler detected an attempt to instantiate a data type that does not require use of the keyword **new**.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i = new int(5);  
        // error: cannot use 'new' on 'int' types  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0109

Cannot 'new' an instance of abstract class '*identifier*'

The compiler detected an attempt to instantiate a class object declared as **abstract**.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        Simple2 s2Object = new Simple2();  
        // error: class 'Simple2' declared as abstract  
    }  
}  
  
abstract class Simple2 {  
    // do something meaningful  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0110

Cannot 'new' an interface '*identifier*'

The compiler detected an attempt to instantiate an interface object declared as **abstract**. Note that interfaces are abstract by default, regardless whether the keyword **abstract** is used in their declaration.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        Simple2 s2Object = new Simple2();  
        // error: interface Simple2 is abstract  
    }  
}  
  
interface Simple2 {  
    // do something meaningful  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0111

Invalid use of array initializer

The compiler detected an attempt to initialize an array, but the initialization statement was not syntactically correct.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0112

Cannot assign final variable '*identifier*'

The compiler detected an attempt to change the value of a variable declared as **final**.

The following sample illustrates this error:

```
public class Simple {  
    private final int i = 3;  
  
    public void method1(int arg1) {  
        i = arg1;  
        // error: variable 'i' declared final  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0113

Call to constructor must be first in constructor

The compiler detected a constructor called from within the body of a second constructor, but the constructor call was not placed at the beginning of the second constructor body.

The following sample illustrates this error:

```
public class Simple {  
  
    int i, j;  
    Simple () {  
  
        i = 0;  
    }  
  
    Simple(int arg1) {  
  
        j = arg1;  
        this(); // error: call to Simple() must be first  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0114

Cannot reference 'this' in constructor call

The compiler detected a reference to **this** in a constructor. As the object has not been fully created while in the constructor, references to **this** are illegal in constructors.

The following sample illustrates this error:

```
class SuperSimple {  
    SuperSimple(Object o) { }  
}  
  
public class Simple extends SuperSimple {  
    Simple()  
    {  
        super(this);  
        //error: cannot refer to this in constructor  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0115

Cannot call constructor recursively

The compiler detected a recursive constructor call.

The following sample illustrates this error:

```
public class Simple {  
    Simple (int arg1) {  
        this(1);  
        // error: constructor calling itself  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0116

Variable '*identifier*' may be used before initialization

The compiler detected an attempt to use a variable before it was properly initialized.

The following sample illustrates this error:

```
public class Simple {  
  
    static  
    {  
        int i;  
        int j = i;  
        // error: 'i' not yet initialized  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0117

Cannot declare a class to be 'private'

The compiler detected use of the modifier **private** in a class declaration. This modifier may only be used with variables and methods.

The following sample illustrates this error:

```
private class Simple {  
    // error: a class cannot be 'private'  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0118

Too many local variables in method - must be <= 256

The compiler detected more than 256 local variables defined within a method. Try reducing the number of variables local to the method and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0119

Too much code in method

The compiler could not support the number of instructions local to a particular method. Try reducing the code in your larger methods by dividing them into smaller, more general purpose methods, and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0120

Divide or mod by zero

The compiler detected a division by zero error in code.

The following sample illustrates this error:

```
public class Simple {  
  
    final int x = 0;  
    int y = 1 % x;  
    //error: x cannot be 0  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0121

Unable to recover from previous error(s)

The compiler encountered a serious error and could not continue processing the file reliably. Try fixing whatever errors are already flagged and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0122

Exception '*identifier*' not caught or declared by '*identifier*'

The compiler detected an exception that was thrown but never caught within the exception class.

The following sample illustrates this error.

```
class SimpleException extends Exception {  
    // do something meaningful  
}  
  
class Simple {  
  
    void method1() throws SimpleException { }  
    void method2() { method1(); }  
    // error: exception not declared for method2  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0123

Multiple inheritance of classes is not supported

The compiler detected a class attempting to apply the keyword **extends** to more than one base class. This is defined as multiple inheritance in other languages, and is not supported in Java.

The following sample illustrates this error:

```
public class Simple extends BaseClass1, BaseClass2 {  
    // error: Multiple inheritance not supported in Java  
}  
  
class BaseClass1 {  
    // do something meaningful  
}  
  
class BaseClass2 {  
    // do something meaningful  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0124

Operator cannot be applied to '*identifier*' values

The compiler detected an operator being applied to a type it cannot be used with.

The following sample illustrates this error:

```
public class Simple {  
    void method1(boolean b) {  
        b++;  
        // error: post increment operator cannot  
        // be applied to boolean variables  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0125

'finally' without 'try'

The compiler detected a **finally** block but did not find a corresponding **try** statement.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        finally {  
            // error: missing corresponding 'try'  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0126

'catch' without 'try'

The compiler detected a **catch** block but did not find a corresponding **try** statement.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        catch {  
            // error: missing corresponding 'try'  
        }  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0127

'else' without 'if'

The compiler detected the keyword **else** but did not find a corresponding **if** statement.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        else // error: no corresponding 'if'  
            ;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0128

Cannot declare an interface to be 'final'

The compiler detected an interface declared with the keyword **final**.

The following sample illustrates this error:

```
final interface Simple {  
  
    // error: 'final' only applies to  
    // classes, methods or variables  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0129

Cannot declare a class to be 'abstract' and 'final'

The compiler detected a class declared with the keywords **abstract** and **final**.

The following sample illustrates this error:

```
public abstract final class Simple {  
  
    // error: 'abstract' and 'final' cannot  
    // be used together in a class declaration  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0130

Cannot declare an interface method to be 'native', 'static', 'synchronized' or 'final'

The compiler detected one of the keywords shown above used in the declaration of an interface method.

The following sample illustrates this error:

```
interface Simple {  
  
    public final void method1();  
    // error: 'method1' cannot be declared  
    // as final in an interface  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0131

Cannot declare a method to be '*identifier*' and '*identifier*'

The compiler detected the use of two or more incompatible modifiers in the declaration of a method.

The following sample illustrates this error:

```
public class Simple {  
    public private void method1() {  
        // error: modifiers 'public' and 'private'  
        // cannot be combined in a declaration  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0132

Cannot declare a field to be '*identifier*' and '*identifier*'

The compiler detected the use of two or more incompatible modifiers in the declaration of a variable.

The following sample illustrates this error:

```
public class Simple {  
    public private int i;  
    // error: modifiers 'public' and 'private'  
    // cannot be combined in a declaration  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0133

Constructors cannot be declared 'native', 'abstract', 'static', 'synchronized', or 'final'

The compiler detected the use of one of the modifiers shown above in the declaration of a constructor.

The following sample illustrates this error:

```
public class Simple {  
  
    final Simple() {}  
    // error: constructors cannot be 'final'  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0134

Interfaces cannot have constructors

The compiler detected an interface containing a constructor declaration.

The following sample illustrates this error:

```
interface Simple {  
  
    Simple();  
    // error: interfaces cannot  
    // declare constructors  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0135

Interface data members cannot be declared "transient" or 'volatile'

The compiler detected one of the modifiers shown above used in the declaration of a interface member variable.

The following sample illustrates this error:

```
interface Simple {  
    volatile int i = 1;  
    // error: 'volatile' cannot be used.  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0136

Public class '*identifier*' should not be defined in '*identifier*'

The compiler detected more than one class declared with the modifier **public** in a source file.

The following sample illustrates this error:

```
public class Simple {  
    // do something meaningful  
}  
  
public class Errorclass {  
    // error: only one class may be defined as  
    // 'public' within the same source file  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0137

Code page '*identifier*' not supported

The compiler detected an unsupported system code page. Refer to your Operating System manual for instructions on changing the system code page.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0138

Interface cannot have static initializer

The compiler detected a **static** initializer within an interface.

The following sample illustrates this error:

```
interface Simple {  
    static {  
        // error: static initializers cannot  
        // be used in interfaces  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0139

Invalid label

The compiler detected an invalid label.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        123:  
        // error: label cannot  
        // begin with alphanumeric  
        return;  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0140

Cannot override static method '*identifier*'

The compiler detected an attempt to override a **static** method from within a subclass.

The following sample illustrates this error:

```
public class Simple {  
    static void method1() {}  
}  
  
class SimpleSubclass extends Simple {  
    void method1() {}  
    // error: cannot override  
    // static method  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0141

Argument cannot have type 'void'

The compiler detected a method argument defined as type **void**.

The following sample illustrates this error:

```
public class Simple {  
    public void method1(void i) {  
        // error: type void can only  
        // be used as a return value  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0142

Cannot make direct (non-virtual) call to abstract method '*identifier*'

The compiler detected an attempt to directly call an abstract method.

The following sample illustrates this error:

```
abstract class Simple {  
    abstract int method1();  
}  
  
class SimpleSubclass extends Simple {  
    int method1() {  
        return super.method1();  
        // error: 'method1' must be  
        // implemented instead  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0143

Cannot throw exception '*identifier*' from initializer '*identifier*'

The compiler detected an attempt to throw an exception from within a static initializer.

The following sample illustrates this error:

```
public class Simple {  
  
    static {  
        ThrowClass TClass = new ThrowClass();  
        // error: cannot throw exceptions  
        // within static initializers  
    }  
}  
  
class ThrowClass {  
  
    ThrowClass() throws Exception{}  
  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0144

Cannot find definition for interface '*identifier*'

The compiler could not locate a definition for the named interface.

The following sample illustrates this error:

```
public class Simple implements Bogus {  
    // error: the interface 'Bogus' does not exist  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0145

Output directory too long: '*identifier*'

The output directory exceeded 228 characters in length. Try shortening the length of the output directory path and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0146

Cannot create output directory '*identifier*'

The output directory could not be created. This error most likely occurs when you do not have write permission on the specified drive.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0147

Cannot access private member '*identifier*' in class '*identifier*' from '*identifier*'

The compiler detected an invalid attempt to access a private member contained within another class.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        AccessClass ac = new AccessClass();  
        ac.i = 1;  
        // error: cannot access 'i'  
    }  
}  
  
class AccessClass {  
    private int i = 0;  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0148

Cannot reference instance method '*identifier*' before superclass constructor has been called
The compiler detected an attempt to reference an instance method before the superclass constructor was called.

The following sample illustrates this error:

```
abstract class Simple {  
    Simple(int i) {}  
    int method1() {  
        return 0;  
    }  
}  
  
class SimpleSubclass extends Simple {  
    SimpleSubclass() {  
        super(method1());  
        // error: constructor must be called first  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0149

The value '*identifier*' cannot be represented by type '*identifier*'

The compiler detected an invalid conversion during an assignment.

The following sample illustrates this error:

```
public class Simple {  
    char c = 65536;  
    // error: char type only supports  
    // >= 0 and <= 65535  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0150

Cannot have repeated interface 'identifier'

The compiler detected an interface name being repeated within a class declaration.

The following sample illustrates this error:

```
interface SimpleI {  
    // do something meaningful  
}  
  
class Simple implements SimpleI, SimpleI {  
    // error: 'SimpleI' repeated  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0151

Variable '*identifier*' is already defined in this method

The compiler detected two variables with the same name defined twice within the same scope of a method.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        int i = 1;  
        int i = 0;  
        // error: 'i' defined twice within  
        // the same scope  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0152

Ambiguous reference to '*identifier*' in interfaces '*identifier*' and '*identifier*'

The compiler detected an ambiguous reference to an identifier. The identifier may have been declared in two or more interfaces, and the compiler could not determine which reference to use.

The following sample illustrates this error:

```
interface Interface1 {
    final int i = 0;
}

interface Interface2 {
    final int i = 1;
}

public class Simple implements Interface1, Interface2 {

    int method1() {
        return i;
    }
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0153

Could not load type library '*identifier*' -- LoadTypeLib() failed

The compiler failed to load the specified type library. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The LCID (locale identifier) could not be found in the OLE support DLLs.
- The type library or DLL could not be loaded.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0154

Could not get library attribute

The compiler failed to retrieve a type library's attributes. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0155

Could not get library name

The compiler failed to retrieve the type library's name. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0156

Could not load library '*identifier*' looking for '*identifier*'

The compiler failed to load the specified library. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0157

Could not load type '*identifier*' from library '*identifier*'

The compiler could not load the specified type from the library shown. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0158

Class '*identifier*' already defined

The compiler detected two or more classes defined with the same name.

The following sample illustrates this error:

```
public class Simple {  
    // do something meaningful  
}  
  
class Simple {  
    // error: class 'Simple' already defined  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0159

'@' must be followed by the response file name

The compiler detected the @ character on the JVC command line, but did not detect a valid response file name immediately following it. Supply the response file name and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0160

response file '*identifier*' could not be opened

The compiler could not open the specified response file. This error most likely occurs when the response file name is misspelled or the file does not exist.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0161

Cannot open source file: '*identifier*'

The source file specified in the error message could not be opened. This error most likely occurs when either the file name specified is misspelled or the file does not exist.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0162

Failed to initialize compiler - maybe you didn't set the class path?

The compiler failed to properly initialize. Check to ensure the **CLASSPATH** variable is set properly and compile again.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0163

Array '*identifier*' missing array index

The compiler detected access to an array type, but the index value was missing.

The following sample illustrates this error:

```
public class Simple {  
    int j[] = {1, 2, 3};  
  
    void method1() {  
        j[] = 0;  
        // error: 'j' missing index value  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0164

Ambiguous import of class '*identifier*' from more than one package

The compiler detected two or more **import** statements attempting to import identical class names from different packages.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0165

Cannot throw exception '*identifier*' from method '*identifier*' -- it is not a subclass of any exceptions thrown from overloaded method '*identifier*'

The compiler detected an override method attempting to throw more exceptions than the method it overrides. In Java, an override method may not be declared to throw more exceptions than the overridden method.

The following sample illustrates this error:

```
class ExceptionA extends Exception {
    // do something meaningful
}

class ExceptionB extends Exception {
    // do something meaningful
}

class AnotherClass {

    public void method1() throws ExceptionA {
        // do something meaningful
    }
}

public class Simple extends AnotherClass {

    public void method1() throws ExceptionA, ExceptionB {
        // error: cannot throw greater than
        // one exception here
    }
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0166

Cannot access member '*identifier*' in class '*identifier*' from '*identifier*' -- it is in a different package

The compiler detected an invalid attempt to reference a member variable defined within a different package. This error most likely occurs when an attempt is made to access a **protected** member defined within another package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0167

Non static methods can't be overridden by a static method

The compiler detected an attempt to override a superclass method with a subclass method declared with the modifier **static**.

The following sample illustrates this error:

```
public class Simple {  
    public void method1() {  
        // do something meaningful  
    }  
}  
  
class Simple2 extends Simple {  
    static public void method1() {  
        // error: overriding superclass 'method1'  
        // with a static method is not valid  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0168

The declaration of an abstract method must appear within an abstract class

The compiler detected a method declared with the modifier **abstract** within a class which was not defined as **abstract**.

The following sample illustrates this error:

```
public class Simple {  
    abstract void method1();  
    // error: class must also be abstract  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Error J0169

Cannot access '*identifier*' -- only public classes and interfaces in other packages can be accessed

The compiler detected an attempt to access a non-public class or interface contained within another package. Only classes or interfaces defined with the modifier **public** can be accessed in other packages.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0170

Cannot load predefined class '*identifier*' -- is CLASSPATH set correctly?

The compiler attempted to load a predefined class, but was unable to find the appropriate file. This error most likely occurs when the **CLASSPATH** variable has not been set correctly.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0171

Could not load type '*identifier*' from library '*identifier*'

The compiler could not load the specified type from the library shown. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0172

Could not load type '*identifier*' from library '*identifier*'

The compiler could not load the specified type from the library shown. This error most likely occurs in any of the following situations:

- The system is low on available memory.
- The path the library resides on does not provide adequate read or write permissions to the user.
- The library format is older and unsupported.
- The library could not be opened.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Error J0173

Found class '*identifier*' in package '*identifier*' rather than package '*identifier*'

The compiler found the specified class, but the class was not defined as a member of the correct package. This error most likely occurs when an **import** statement includes an incorrect package identifier for the class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5001

Local variable '*identifier*' is initialized but never used

The compiler detected an initialized variable that was never referenced in any class code. This message occurs at warning level 3 or greater.

The following sample illustrates this warning:

```
public class Simple {  
    public int method1() {  
        int i = 1;  
        return 1;  
        // warning: 'i' is never used  
    }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler Warning J5002

Compiler option '*identifier*' is not supported

The compiler detected an unsupported command line option specified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5003

Ignoring unknown compiler option '*identifier*'

The compiler detected an unknown option specified on the JVC command line. This warning most likely occurs when a typographical error exists. For example, specifying ***/W4*** on the command line will cause this warning because the **warning level** option must use a lower-case 'w'.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5004

Missing argument for compiler option '*identifier*'

The compiler detected a valid command line option, but the required argument was not specified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5005

Package '*identifier*' was already already implicitly imported

The compiler detected an **import** statement for a package that was already implicitly imported, such as **java.lang**. This message occurs at warning level 1 or greater.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5006

'private protected' not supported, using 'protected'

The compiler detected use of the modifier combination **private protected**. This combination is now obsolete and has been replaced by **protected**. This message occurs at warning level 1 or greater.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5007

Non standard conversion from '*identifier*' to '*identifier*'

The compiler detected an assignment conversion problem that, while invalid according to the current Java language specification, must be supported for compatibility with the Sun class libraries. This message occurs at all warning levels.

The following sample illustrates this warning:

```
public class Simple {  
    public void method1() {  
        short s = 5;  
        int    i = 50000;  
        long   l = 50000000;  
  
        s += i; // warning: non-standard assignment  
        s += l; // warning: same as above  
        i += l; // warning: same as above  
    }  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5008

Method '*identifier*' contains data types that cannot be accurately and safely represented in Java. The method will not be converted

JavaTLB detected a method with a parameter or return value that cannot be safely represented in Java.

Some of the data types that can generate this error are:

- parameters that are complex or nested structures
- parameters having multiple levels of pointer indirection
- return values (or parameters marked with the **retval** attribute) that are array types

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler Warning J5009

Method '*identifier*' is incompatible with Automation

JavaTLB detected a method with a parameter or return value that is not supported for a dispatch interface. This includes parameters or return values of type LPSTR or LPWSTR. (Automation interfaces typically use the BSTR data type to represent strings.)

If the method was declared in a dual interface, the method is converted so that you can call the v-table version. If the method was declared in a dispatch-only interface, the method is not converted.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Acknowledgments

Many of the definitions included in this glossary are published in the *Microsoft Press Computer Dictionary*, Copyright 1994 and are reprinted with the permission of the publisher, Microsoft Press.

The Microsoft Press Computer Dictionary is available for purchase by calling 1-800-MSPRESS. Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office. Or contact Microsoft Press International directly at fax (206) 936-7329.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Glossary - Nonalphabetic Terms

& (ampersand)

- 1 Bitwise-AND operator. Two ampersands (**&&**) denote the logical-AND operator.
- 2 In C++, the reference operator, as in `void AddIt(int&x)`. This indicates that the address of `x` is used directly, and changes in value affect the original variable. Java does not allow reference operators. See also parameter passing.
- 3 In C/C++, the address-of operator, as in `pPtr = &nArray[2]`. Java does not allow address-of operators.
- 4 Indicates special characters in HTML code.

.APS

A binary version of the current resource file that is created by the Microsoft Developer Studio and used for quick loading of resources. Microsoft Developer Studio gives this file an .APS filename extension.

.BAT

An unformatted text file that contains one or more commands, either internal operating-system commands or program names. A batch file is executable and can be run from the command line.

.BMP

A file that contains a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

Bitmap files usually have a .BMP filename extension. See also bitmap, device-independent bitmap file.

.BSC

A file created from source browser information (.SBR) files, using the Microsoft Browse Information File Maintenance Utility (BSCMAKE). Browse information files can be examined in browse windows and usually have a .BSC extension.

.bss

A predefined data section of an executable file that contains uninitialized data, including all variables declared as static within a function or source module. The linker combines all the .bss sections in the object (.OBJ) and library (.LIB) files into one .bss section in the executable file.

.C

A text file containing C language code.

.CLW

A file that ClassWizard generates, containing information needed to edit existing classes or add new classes to a project. ClassWizard also uses the ClassWizard file to store information needed to create and edit message maps and dialog data maps, and to create prototype member functions. ClassWizard files have a .CLW filename extension.

.COM

An executable binary (program) file whose code is limited to a single 64-kilobyte segment. Compact executable files usually have a .COM filename extension and are often used for utility programs and short routines. See also executable file.

.CPP

Or .CXX file. A text file containing C++ source code.

.CUR

A file that contains an image that defines the shape of a cursor on the screen. Cursor resource files usually have a .CUR filename extension.

.DEF

A text file that contains one or more statements describing various attributes of an executable module. Module-definition files usually have a .DEF filename extension. See also dynamic-link library (.DLL) file.

.DIB

A file containing an array of bits combined with several structures that specify the width and height of the bitmapped image (in pixels), the color format of the device where the image was created, and the resolution of the device used to create that image. The DIB file format ensures that bitmap graphics created in one application can be loaded and displayed in another application exactly the way they appear in the originating application. See also bitmap, bitmap file.

.DLG

A file that contains dialog-box source code. Note that a dialog file is not required for a Visual C++ project because Visual C++ keeps this code in the resource-definition file.

.DLL

A file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. In Win32, the operating system maps the dynamic-link libraries (DLLs) into the address space of a process when the process is starting up or while it is running. The process then executes functions in the DLL. Dynamic-link library files usually have a .DLL filename extension.

.EXE

A program file created from one or more source code files translated into machine code and linked together. The MS-DOS, Windows, and Windows NT operating systems use the .EXE filename extension to indicate that the file is a runnable program.

.EXP

A file that contains information about exported functions and data items. The Microsoft 32-Bit Library Manager tool (LIB.EXE) generates the exports file from the module-definition (.DEF) file. The linker

uses the exports file to build the dynamic-link library (.DLL) file. Exports files have a .EXP filename extension.

.H

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

.HLP

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

.HM

A file that defines Help context IDs corresponding to the IDs of dialog boxes, menu commands, and other resources in an application. The AppWizard file MAKEHELP.BAT calls the MAKEHM tool to generate this file from the contents of a RESOURCE.H file. The Help map file has a .HM filename extension. See also Help project file.

.HPJ

A project file that controls how the Windows Help Compiler creates a Help (.HLP) file from topic files. The Microsoft Help Workshop is used to create a Help project file. The filename extension of a Help project file is .HPJ.

.HPP

Or .HXX file. An external source file, identified at the beginning of a C++ program, that contains commonly used data types and variables used by functions in a program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program.

.ICO

In Windows, a file that contains a bitmap of an icon. Icon files usually have a .ICO filename extension.

.ILK

A state file generated to hold status information for later incremental links of the program. The file has the same base name as the executable file or dynamic-link library and the filename extension .ILK. The incremental status file is created the first time the Incremental Linker (LINK.EXE) runs in incremental mode. LINK updates the file during subsequent incremental builds. LINK is the only tool that uses the .ILK file. See also incremental link.

.INI

In Windows, a file that an application uses to store information that otherwise would be lost when the application closes. Initialization files typically contain information such as user preferences for the configuration of the application. Initialization files usually have a .INI filename extension.

.LIB

A Common Object File Format (COFF) file generated by the Microsoft 32-bit library manager tool, LIB, for standard and import libraries. The default filename extension for these files is .LIB. See also dynamic-link library (.DLL), static-link library.

.MAK

A file that contains all commands, macro definitions, options, and so on to specify how to build the projects in a project workspace. A makefile has the filename extension .MAK and usually has the same base name as the workspace configuration (.MDP) file.

.MAP

A text file that contains information about the program being linked, including the groups in the program and a list of public symbols. The linker names the mapfile with the base name of the program and the filename extension .MAP.

.MDP

A file created and maintained by the Developer Studio. The file contains all the information about the project workspace, including a list of all the projects, their associated source files, each project's output file, the settings and tools required to build each project, as well as the physical layout and characteristics of Developer Studio: window layout and characteristics, syntax coloring, editor preferences, and so on.

.OBJ

A file containing object code and/or data generated by a compiler or an assembler from the source code of a program. Object files generated by the Visual C++ compiler have a .OBJ filename extension. See also Common Object File Format (COFF).

.ODL

In OLE Automation, text files containing a description of an application's interface. Object description language scripts are compiled into type libraries using the MktypLib tool included with the OLE Software Development Kit.

.OLB

A dynamic-link library with a type library resource. An object library file typically has a .OLB filename extension.

.PBI

In a profiling operation, a file that provides condensed information to the Visual C++ profiler (PROFILE). The PREP program generates a profiler batch input file the first time the profiler is run on a program. The default filename extension for profiler batch input files is .PBI. See also profiler batch output file, profiler batch text file.

.PBO

An intermediate file generated by the Visual C++ profiler (PROFILE) and used to transfer information between profiling steps. See also profiler, profiler batch input file, profiler batch text file.

.PBT

In a profiling operation, the file generated by the PREP program and used as input to the PLIST program to generate a human-readable profile of the source code. See also profiler, profiler batch input file, profiler batch output file.

.PCH

A file containing compiled code for a portion of a project. Subsequent builds combine this file with the

uncompiled code, thus shortening the overall compile time. The default filename extension for a precompiled header file is .PCH.

.PDB

A file used by the build tools to store information about a user's program. The program database file speeds linking during the debugging phase of development by keeping the debugging information separate from the object files.

.RC

Or resource script file. A text file containing descriptions of resources from which the resource compiler creates a binary resource file. For Microsoft Windows applications, resource-definition files usually have a .RC filename extension. For Apple Macintosh applications, such files are typically named with a .R extension and written with the Apple Rez script language. See also compiled resource (.RES) file.

.REG

In OLE applications, a text file description of the classes supported by a server application. When a server application is installed in a system, the contents of its registration entry file are merged with the system registry. Registration entry files usually have a .REG filename extension.

.RES

Or binary resource file. A binary file that contains a Windows-based application's resource data and is created by the resource compiler from the resource-definition (.RC) file. Compiled resource files usually have a .RES filename extension. See also Macintosh binary resource file, resource compiler.

.RSC

A Macintosh resource file that has been created from a Windows resource script (.RC) and compiled using the Windows Portability Library version of the Windows Resource Compiler (RC.EXE). By default, .RSC is the filename extension for Macintosh binary resource files.

.RTF

A file that contains encoded, formatted text and graphics for easy transfer between applications. The rich-text encoding format is commonly used by document-processing programs such as Microsoft Word for Windows and for generating online Help files. Rich-text format files usually have a .RTF filename extension.

.SBR file

An intermediate file that the compiler creates for use by the Microsoft Browse Information Maintenance Utility (BSCMAKE). There is one .SBR file for each object (.OBJ) file. BSCMAKE uses the .SBR files to create a browse information (.BSC) file.

.TLB

Or OLE library. An OLE compound document file containing standard descriptions of data types, modules, and interfaces that can be used to fully expose objects for OLE Automation. The type library file usually has a .TLB filename extension and can be used by other applications to get information about the automation server.

.TXT

A human-readable file composed of text characters. A text file is usually identified by a file extension of .TXT. See also binary file, rich-text format file.

.WAV

A Microsoft standard file format for storing waveform audio data. Wave files have a .WAV filename extension.

.WRI

A document file that is associated with the Windows Write text editor. The default filename extension for Windows Write files is .WRI

@ (at sign)

- 1 In e-mail addresses, separates the user name from the domain name. See also address.
- 2 Points to the input command file used by CL or LINK, as in `LINK @LINK.RSP`. See also command file.

16-bit application

A program written for a system (such as MS-DOS or Microsoft Windows version 3.1) that uses a 16-bit segmented architecture, in which each memory address points to a 16-bit word.

16-bit character

A character that is 2 bytes in size, unlike an ANSI character, which is 1 byte in size. Sixteen-bit characters are found in Unicode sets, multibyte character sets (MBCS), and double-byte character sets (DBCS).

32-bit application

A program written for a system (such as Microsoft Windows NT or Windows 95) that uses a 32-bit architecture, in which each memory address points to a 32-bit word.

8.3 filename convention

The naming convention for filenames in MS-DOS that allows up to eight characters, with an optional period and three-character extension. See also base name, filename extension.

Glossary A

A5-relative reference

An address designated by a 16-bit offset from the A5 register in the 680x0-based Apple Macintosh. An A5-relative reference is similar to the near data address in Intel-based machines.

ABC character spacing

See ABC width.

ABC width

Or ABC character spacing. The amount of spacing required for a single glyph in a font. A spacing is added to the current position before drawing the glyph. B spacing is the width of the black part of the glyph. C spacing is added to the current position to account for the white space to the right of the glyph. The total advanced width is given by A+B+C. See also overhang, underhang.

abnormal termination

In exception handling, the condition that occurs when a program leaves the **try** block of a **try-finally** statement before the code executes to the closing brace. For example, statements such as **return**, **goto**, **continue**, and **break** can cause abnormal termination.

absolute symbol

A symbol that contains a constant value for an address in memory not associated with a program address.

absolute time

An expression of time that stays the same regardless of the time zone.

abstract base class

See abstract class.

abstract class

Or abstract base class. In C++, a class that cannot be instantiated but is used as a base from which other classes can be derived. An abstract class contains at least one pure virtual function; if its derived classes do not implement these pure virtual functions, then they, too, become abstract classes.

accelerator key

Or keyboard accelerator, shortcut key, keyboard shortcut. A keystroke or combination of keystrokes that invokes a particular command. See also accelerator table, access key.

accelerator resource

See accelerator table.

accelerator table

Or accelerator resource. A data structure that contains a list of accelerator keys and the command identifiers associated with them.

access key

Or mnemonic key. The key that corresponds to an underlined letter on a menu or dialog-box item. An access key can be used to invoke the command associated with that menu item or dialog-box item. See also accelerator key.

access mode

A mode that determines the manner in which entry to a file, directory, or other data source object is permitted, as well as the manner in which changes are made to the data. See also direct mode, transacted mode.

access privileges

In object-oriented programming, the degree to which a class grants outside access to its data and functions. See also friend.

access token

A group of security attributes permanently attached to a process when a user logs on to the operating system. An access token contains privileges and security identifiers for a user, global group, or local group. The privileges regulate the use of some system services and the security identifiers regulate access to objects that are protected by access-control lists (ACLs). There are two kinds of access tokens: primary and impersonation. See also impersonation token, primary token, privilege, security identifier (SID).

access-control list

A list of security protections that applies to an object (a file, process, event, or anything else having a security descriptor). An entry in an ACL is an access-control entry (ACE). There are two types of access-control lists: discretionary and system. See also security descriptor.

ACL

A list of security protections that applies to an object (a file, process, event, or anything else having a security descriptor). An entry in an ACL is an access-control entry (ACE). There are two types of access-control lists: discretionary and system. See also security descriptor.

activation

The process of making a window or other object operational. Activation of a window, for example, changes the color of the title bar, moves the window to the top of the Z order, and enables keyboard focus. In OLE, the term activation is often used to refer to in-place activation of embedded objects.

active HTML document

A Web page that contains ActiveX controls, active scripts, or Java applets.

active state

1 In OLE, the state of an OLE object in a compound document when the user can edit the embedded object without leaving the container document's window.

2 In general, the state of a program, document, device, or portion of the screen that is currently operational.

ActiveX

All component technologies built on Microsoft's Component Object Model (COM), other than Object Linking and Embedding.

ActiveX control

The new name for programmable elements formerly known variously as OLE Controls, OCXs, or OLE Custom Controls. Controls previously built with the MFC Control Developer's Kit meet the ActiveX control specification.

ActiveX document

A document that contains ActiveX controls, Java applets, or ActiveX document objects. Also called active object or active script.

ActiveX scripting

Microsoft technology for connecting third-party script engines to applications.

ActiveX server extension

A dynamic-link library (DLL) that creates server extensions on any ISAPI-compliant Web server.

ActiveX server filter

A dynamic-link library (DLL) that intercepts and processes notifications directed to any ISAPI-compliant Web server.

ActiveX server framework

Microsoft tools for creating extensions to ISAPI-compliant Web server software. See also ActiveX server extension, ActiveX server filter, ActiveX scripting.

actual argument

1 Or actual parameter. A unit of information (variable, constant, pointer, etc.) passed in a macro call or inside the parentheses of a function call that the called function then converts to its formal parameter.

2 In the Visual C++ documentation, the argument passed to a C++ template class.

actual parameter

See actual argument.

actual parameter list

The arguments specified in a particular method or function call. See also formal parameter list.

address

The path to a value, object, document, page, or other destination. An address can be a memory location, a URL (address to an Internet site) or a UNC network path (address to a file on a local area network). An address may also contain more specific information such as a database object, a bookmark in a file, or a spreadsheet cell range to which the main address points.

address space

1 Or memory space. The portion of memory allocated to a given process. See also heap.

2 Or memory space. More generally, the range of memory locations to which a microprocessor can refer. Effectively, a computer's address space is the amount of memory a microprocessor could use if all the memory was available.

address-of (&) operator

In C/C++, a unary operator that gives the address of its operand, which can be either a function name or an l-value. The result of the address-of operation is a pointer to the operand. The type addressed by the pointer is the type of the operand. Note that the ampersand (&) character is also used by the bitwise-AND (&) operator. See also bitwise operator, indirection (*) operator.

AFXDLL

Redistributable dynamic-link libraries (DLLs) containing the entire 32-bit Microsoft Foundation Class Library. AFXDLL enables you to build an application without statically linking to the MFC object-code libraries. The architecture is useful in building MFC extension DLLs and in sharing the class library between multiple executable files, thus saving disk space and memory.

aggregate object

In the Component Object Model, an object whose implementation of certain interfaces is provided by one or more of its contained objects.

aggregate type

Or complex type, nonscalar type. In C/C++, a structure, union, or array data type. In C++, a class can also be an aggregate type, provided it does not have constructors, nonpublic members, base classes, or virtual functions.

aliasing

1 In programming, to use an alternate name to refer to a memory location that is already referred to by a different name, or to allow two pointers to point to the same memory location. The alternate name is an alias.

2 In computer graphics, a rendering technique that assigns to pixels the color of the primitive being rendered, regardless of whether that primitive covers all or only a portion of the pixel's area. This results in a jagged, or stairstep, appearance of certain design elements, such as diagonal lines, curves, and circles.

3 More generally, to substitute one name, or alias, for another name, or for a group of names. For example, a long Macintosh filename could be aliased with an MS-DOS 8.3-format filename.

allocation

See memory allocation.

alpha value

In mixing models, the component used to control color blending. See also red, green, blue, alpha (RGBA).

ambiguity

1 In derived classes, a condition that occurs when an expression can refer to more than one data type, object, or function. Use of the scope resolution (::) operator resolves the ambiguity.

2 In syntax, a condition that can occur with a type cast, especially a function-style type cast. For

example, it is unclear whether the expression `char *aName(String(s))` is a function declaration or an object declaration with a function-style cast as the initializer.

3 More generally, two or more meanings for a single expression, which is anathema to a binary system.

ambiguous expression

1 An expression whose value depends on a particular order of evaluation where the language does not define one. For example, the values received by `func(i, ++i)` depend on whether its parameters are passed from right to left or from left to right.

2 An expression that cannot be evaluated because the types used in the expression are not unique.

American National Standards Institute

An organization of American industry and business groups dedicated to the development of trade and communication standards. ANSI sets standards for C and other programming languages to eliminate variations that could cause problems in transporting a program from one type of computer system or environment to another.

American Standard Code for Information Interchange

The dominant standard for coding information on computers and related equipment. The ASCII coding scheme assigns numeric values to letters, numbers, punctuation marks, and certain other characters, enabling computers and computer programs to exchange information. See also ANSI character set, Unicode.

anchor

The HTML element that connects Web documents. Anchors either jump to another location, or are jumped to by other anchors. They are similar in function to bookmarks.

anonymous union

In C++, a union without a tag or declarators. An anonymous union declares an unnamed object and cannot have member functions or private or protected members.

ANSI

An organization of American industry and business groups dedicated to the development of trade and communication standards. ANSI sets standards for C and other programming languages to eliminate variations that could cause problems in transporting a program from one type of computer system or environment to another.

ANSI c

The American National Standards Institute's 1990 version of the C language, which specifies the formal syntax and semantics of the language, the standard libraries that are included, and the way in which the compiler is to translate the program. ANSI compliance promotes portability of programs across different platforms.

ANSI character set

An 8-bit character set that contains the 7-bit ASCII standard character set as well as currency and mathematical symbols, accented characters, and other characters not normally found on the keyboard. Microsoft Windows version 3.1 and its applications use the ANSI character set internally.

See also OEM character set, Unicode.

ANSI string

A string composed of characters from the ANSI character set.

API

A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system. For computers running a graphical user interface, an API manages an application's windows, icons, menus, and dialog boxes.

App Studio precompiled file

A binary version of the current resource file that is created by the Microsoft Developer Studio and used for quick loading of resources. Microsoft Developer Studio gives this file an .APS filename extension.

Apple Shared Library Manager

A library of code and resources that can be used simultaneously by more than one Macintosh application. (ASLM is not supported on the Power Macintosh.) See also stand-alone code.

applet

- 1 An HTML-based program that the browser temporarily downloads to a user's hard drive, where it runs when the Web page is open. Most often, applet refers to programs developed with Java.
- 2 Any small application.

AppleTalk

A simple local area network developed by Apple Computer that can be used by both Apple and non-Apple computers for communicating and sharing resources such as printers and file servers.

application class

The class, derived from the MFC class **CWinApp**, that encapsulates the initialization, running, and termination of a Windows-based application. An application must have exactly one object of an application class. See also application object.

application framework

Or framework. A group of C++ classes in the Microsoft Foundation Class Library that provides the essential components of an application for Windows. The application framework defines the skeleton, or framework, of an application and supplies standard user-interface implementations that can be placed onto the skeleton. See also class library.

application object

The single instance of the application class. The application object controls documents, views, frame windows, and templates, and specifies application behavior such as initialization and cleanup for every instance of the application.

application programming interface

A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system. For computers running a graphical user interface, an API manages an

application's windows, icons, menus, and dialog boxes.

application queue

In Microsoft Windows, a repository for messages awaiting processing by a particular application. As Windows removes messages from its system message queue, it dispatches the messages that contain application input to the relevant application queue. See also message loop, message queue.

Archie

A tool for finding files stored on anonymous FTP sites. You need to know the exact file name or a substring of it. See also Very Easy Rodent-Oriented Net-wide Index to Computerized Archives (Veronica).

archive

- 1 An object, derived from the MFC class **CArchive**, that provides a type-safe buffering mechanism for writing or reading serializable objects to or from a **CFile object**—a disk file or a memory file (perhaps representing the Clipboard). A given **CArchive object** either stores (writes, serializes) data or loads (reads, deserializes) data, but never both. See also serialization.
- 2 In general, a stored backup copy of data or a program, or the act of storing a backup copy of data or a program.

argument

A value or an expression used with an operator or passed to a subprogram (subroutine, procedure, or function). The program then carries out operations using the arguments. Synonymous with parameter.

arithmetic shift

A shift operation whose left operand is a signed quantity. In a right-shift operation, the sign bit is propagated into the bits vacated by the shift. See also logical shift.

array

An aggregate data type in which all the data items (elements) are of the same type. The elements are accessed by specifying the name of the array and a subscript (index), which represents the element's offset from the base address of the array. For example, `myArray[3]` represents the fourth element in the array. See also subscript (**[]**) operator.

ASCII

The dominant standard for coding information on computers and related equipment. The ASCII coding scheme assigns numeric values to letters, numbers, punctuation marks, and certain other characters, enabling computers and computer programs to exchange information. See also ANSI character set, Unicode.

ASCII character

A character that is part of the American Standard Code for Information Interchange (ASCII). The ASCII character set is a standard 7-bit code for representing characters—letters, digits, punctuation, and control instructions—with binary values.

ASLM

A library of code and resources that can be used simultaneously by more than one Macintosh application. (ASLM is not supported on the Power Macintosh.) See also stand-alone code.

aspect ratio

The ratio of a pixel's width to height on a particular device. Information about a device's aspect ratio is used in the creation, selection, and display of fonts.

assertion

A Boolean statement in the debug version of a program that tests a condition that should, if the program is operating correctly, evaluate as true. If the condition is false, an error has occurred and the program will typically issue an error message that gives the user the option to abort the program, activate the debugger, or ignore the error.

assignment operator

An operator used to assign a value to a variable or a data structure. An example of an assignment operator is the simple assignment (=) operator, which assigns the value of its right operand to its left operand.

assignment statement

A statement that assigns a value to a variable or data structure. An assignment statement is usually composed of a destination variable, an assignment operator, and an expression to be assigned.

associativity

The order (either right to left or left to right) in which expressions are evaluated when adjacent operators have equal precedence and subexpressions are not enclosed in parentheses. For example, the expression $10 < 20 < 5$ evaluates to 1 (true) because the less-than (<) relational operator has left-to-right associativity, causing the expression to be evaluated $(10 < 20) < 5$.

asynchronous operation

- 1 Or overlapped I/O. In programming for Windows, a task that proceeds in the background, allowing the thread that requested the task to continue to perform other tasks. See also synchronous operation.
- 2 More generally, an operation that proceeds independently of any timing mechanism such as a clock.

asynchronous processing

In ODBC, a method of processing transactions in which the database driver returns control to an application before a function call completes; the application can continue nondatabase processing while the driver completes the function in progress. See also synchronous processing.

atomic operation

An operation that never has its execution suspended while partially completed. See also synchronous operation.

audio-video interleaved

A Microsoft technology for displaying full-motion video in a window. An AVI file interleaves waveform audio and video data and has a .AVI filename extension.

authentication

The process of determining the identity of a user attempting to access a system. Note that the user can be a person, a computer, or a process.

authentication token

A portable device used for authenticating a user. Authentication tokens operate by challenge/response, time-based code sequences, or other techniques.

auto storage class

See automatic storage class.

automatic storage class

Or **auto** storage class. In C++, the storage class for objects and variables that are local to the block of code where they are declared. The automatic storage class can be declared explicitly, by using the keywords auto or register, or implicitly, by default. See also local variable, static storage class.

automation client

In OLE Automation, an application that can manipulate exposed objects belonging to another application (called the OLE Automation server). The client drives the server application by accessing the objects' properties and functions. Microsoft Visual Basic is an example of an OLE Automation client application. See also automation server.

AVI

A Microsoft technology for displaying full-motion video in a window. An AVI file interleaves waveform audio and video data and has a .AVI filename extension.

Glossary B

backbone

A high-performance network connecting other networks.

background color

The color of the client area of an empty window or display screen, on which all drawing and color display take place. See also background mode, foreground color.

background mode

A mode that defines how background colors are mixed with existing window or screen colors for bitmap and text operations. The background mode can be either opaque (the default) or transparent.

backward compatibility

- 1 Ensuring that existing applications will continue to work in the new environment.
- 2 Ensuring that the new release of an application will be able to handle files created by a previous version of the product.

bandwidth

The amount of data that can travel through a circuit, usually indicated in bits per second (bps). A measure of capacity, not speed.

bandwidth on demand

In ISDN, the ability to aggregate B channels as the data traffic exceeds preset thresholds.

base address

In relation to memory locations, the portion of a two-part address that remains constant and provides a reference point, or base, from which the location of a byte of data can be calculated. A base address is accompanied by an offset value that is added to the base to determine the exact location (the absolute address) of the information.

base class

In C++, a class from which other classes are derived by inheritance. See also abstract class, derived class, virtual base class.

base line

In the printing and on-screen display of characters, an invisible, horizontal line within a character cell of a given font. Most characters sit on the base line, although some characters, such as *g* and *j*, descend below it.

base name

The portion of the filename that precedes the extension. For example, SAMPLE is the base name of the file SAMPLE.ASM. See also filename extension.

base priority

In multitasking, the priority level that provides the basis for calculating a thread's dynamic priority for CPU time. A thread inherits its base priority from the process in which the thread was created.

The scheduler, which determines the order in which threads should execute, can temporarily boost the priority of a thread, but it cannot reduce the priority below this base priority.

BASED_CODE

A 16-bit MFC macro that ensures that data will be placed in the code segment instead of the data segment. Under Win32, this macro expands to nothing and is provided for backward compatibility.

basic input-output system

A set of routines that work closely with the hardware and the operating system to support the transfer of information between elements of the system, such as memory, disks, and the monitor.

basic rate interface

An ISDN service that consists of two 64-Kbps bearer channels (B channels) and one 16-Kbps signaling channel (D channel) referred to as 2B+D.

batch file

An unformatted text file that contains one or more commands, either internal operating-system commands or program names. A batch file is executable and can be run from the command line.

big-endian

One of two byte-ordering conventions used on different machines. In big-endian addressing, the address points to the most significant byte of the word. Scalar Processor ARChitecture (SPARC) and Motorola 680x0 machines are big-endian machines. See also little-endian.

binary

- 1 The base-2 counting system, whose digits are 0 and 1.
- 2 An executable file—one that is in binary, or machine-readable, form. See also binary file.
- 3 A means of opening a file for input and output (binary mode versus text mode).

binary file

Or binaries. A file that consists of binary data or executable code. For example, a C++ executable file is a binary file. See also text file.

binary large object

- 1 Or binary data object. A large piece of data, such as a bitmap. A BLOB is characterized by large field values, an unpredictable table size, and data that is formless from the perspective of a program.
- 2 A keyword designating the BLOB structure, which contains information about a block of data.

binary operator

An operator that takes two operands. In C, binary operators are the multiplicative operators (*, /, %), additive operators (+, -), shift operators (<<, >>), relational operators (<, >, <=, >=, ==, !=), bitwise operators (&, |, ^), logical operators (&&, ||), the sequential-evaluation operator (,), the assignment operator (=), and the compound-assignment operators (+=, *=, etc.). See also ternary operator, unary operator.

binding

- 1 The association of two pieces of information with one another, most often used in terms of a symbol (such as the name of a variable) with some descriptive information (such as a memory address, a data type, or an actual value).
- 2 In networking, a process that establishes the initial communication channel between the protocol driver and the network adapter card driver.
- 3 In OLE, the process of getting a compound-document object into a running state so that it can be activated. See also activation.
- 4 A Windows Sockets keyword that identifies the **bind** socket library routine. The bind routine associates a local address with a socket.

BIOS

A set of routines that work closely with the hardware and the operating system to support the transfer of information between elements of the system, such as memory, disks, and the monitor.

bit field

A structure member whose width is specified in bits, or binary digits, rather than being specified implicitly as characteristic of the data type. When viewed in binary form, each bit or group of bits in the bit field corresponds to a specific field of information.

bitmap

- 1 Or pixel image, pixel map. An array of bits that contains data describing the colors found in a rectangular region on the screen (or the rectangular region found on a page of printed paper).
- 2 Or bit image. A sequential collection of bits that represents, in memory, an image to be displayed on the screen or printed. See also bitmap file.

bitmap file

A file that contains a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

Bitmap files usually have a .BMP filename extension. See also bitmap, device-independent bitmap file.

bitmap-stretching mode

A mode that defines how information is removed from or combined together in bitmaps that are stretched or compressed. For example, a particular bitmap-stretching mode may preserve black pixels at the expense of colored or white pixels.

bitwise operator

An operator that works on individual bits of its operands. The bitwise-NOT (~) operator produces the bitwise complement of its operand. For example, ~1010 = 0101. The binary operators—bitwise-AND (&), bitwise-OR (|), and bitwise-XOR (^)—compare each bit of the first operand to the corresponding bit of the second operand. For example, 1010 & 0110 = 0010. See also logical

operator.

BLOB

- 1 Or binary data object. A large piece of data, such as a bitmap. A BLOB is characterized by large field values, an unpredictable table size, and data that is formless from the perspective of a program.
- 2 A keyword designating the BLOB structure, which contains information about a block of data.

BOND

In ISDN, the ability to aggregate B channels as the data traffic exceeds preset thresholds.

bookmark

A marker that uniquely identifies a specific record or row in a database, a specific line in SOURCE code, or an item or location in a word-processing file. The HTML equivalent of a bookmark is an anchor with the NAME attribute; this type of anchor is used as a destination for hyperlinks. When creating HTML documents in Word for Windows, for example, you use bookmarks to create anchors with the NAME attribute.

Boolean

A binary algebra that uses the logical operators AND, OR, XOR, and NOT, and whose outcomes consist of logical values (either TRUE or FALSE). The keyword boolean indicates that the expression or constant expression associated with the identifier takes the value TRUE or FALSE. See also conditional expression.

bounding rectangle

Or bounding box. A rectangular area that defines the outer limits of a rounded shape such as an ellipse, arc, or pie.

breakpoint

A location in a program where execution is stopped to allow the developer to examine the program's code, variables, and register values and, as necessary, to make changes, continue execution, or terminate execution.

BRI

An ISDN service that consists of two 64-Kbps bearer channels (B channels) and one 16-Kbps signaling channel (D channel) referred to as 2B+D.

bridge

- 1 A device that connects two segments or components of a network that use the same protocols. Used to overcome limitations on the physical length of a segment. See also brouter, router.
- 2 In ISDN, a hardware device used for forwarding data between two networks. A bridge reads each packet's address information, which is written into the packet at the data-link layer of the OSI model. Unlike a router, a bridge is not interested in the address on the remote LAN, but only in whether or not the packet should be forwarded.

brouter

A device combining the functions of a bridge and a router by connecting two segments of a network and connecting two networks or a network to the Internet. See also bridge, router.

browse buttons

- 1 In Windows Help, a pair of buttons used to browse backwards and forwards through a sequence of Help topics. A browse sequence typically consists of two or more related topics that are intended to be read sequentially.
- 2 In dialog boxes, buttons used to view a list of network servers, directories, files, etc.

browse information file

A file created from source browser information (.SBR) files, using the Microsoft Browse Information File Maintenance Utility (BSCMAKE). Browse information files can be examined in browse windows and usually have a .BSC extension.

browser

A program used to view formatted Web documents.

brush

A graphics object used to paint the interior of shapes and paths. A brush describes an 8- by 8-pixel bitmap. See also logical brush, physical brush.

BSCMAKE

The Microsoft Browse Information File Maintenance utility. BSCMAKE uses source browser information (.SBR) files created by the compiler using the /FR or /Fr option, which are used to create browse information (.BSC) files.

buffer

- 1 (noun) An intermediate repository of data—a reserved portion of memory in which data is temporarily held pending an opportunity to complete its transfer to or from a storage device or another location in memory.
- 2 (verb) To store or collect in a buffer.

build

- 1 (noun) The process of compiling and linking source code to generate an executable program, library, Help file, or other run-time file. The files produced from the build process are also sometimes referred to as a build.
- 2 (verb) To compile and link source code in order to generate an executable program, Help file, or other run-time file.

build tag

A string assigned to a topic that the Help project file can specify to include or not include in a build. Build tags can be made up of any alphanumeric characters and are not case-sensitive. They provide a means of supporting different versions of a Help system without the need to create different source files for each version. Topics without build tags are always included in a build, along with all tagged topics not expressly excluded from that particular build in the build expression.

bulk record field exchange

When bulk row fetching is implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an

external data source. See also record field exchange (RFX), DAO record field exchange (DFX), and dialog data exchange (DDX).

bulk RFX

When bulk row fetching is implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), DAO record field exchange (DFX), and dialog data exchange (DDX).

bulk row fetching

In ODBC, the process of retrieving multiple rows from the data source in a single fetch operation. The number of rows retrieved depends on the recordset object's setting for the rowset size.

button control

A graphical control that enables a user to provide input to an application. Windows provides five kinds of button controls: pushbuttons, check boxes, radio buttons, group boxes, and owner-draw buttons.

byte

A unit of information consisting of 8 bits. A byte, or binary term, is the smallest collection of bits that can be accessed directly.

bytecode

Machine-independent code generated by the Java compiler and executed by the Java interpreter or compiled at the last minute by a JIT compiler.

Glossary C

C calling convention

The C standard for calling a function—that is, pushing arguments onto the stack from right to left (in reverse order from the way they appear in the argument list). After the function returns, the calling function removes the arguments from the stack. The C calling convention permits a variable number of arguments to be passed. See also calling convention.

C linkage specifier

A declaration of a function or object as `extern C`, indicating to the C++ compiler that the function name that follows the linkage specifier is an undecorated C function. C linkage allows existing C code to be used in new C++ applications. See also linkage specification.

C source file

A text file containing C language code.

C++ exception handling

Built-in support provided by the C++ language for handling anomalous situations, known as exceptions, that may occur during the execution of a program. With C++ exception handling, a program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control. See also structured exception handling (SEH).

C++ header file

Or .HXX file. An external source file, identified at the beginning of a C++ program, that contains commonly used data types and variables used by functions in a program. The `#include` directive is used to tell the compiler to insert the contents of a header file into the program.

C++ source file

Or .CXX file. A text file containing C++ source code.

cache

A special memory subsystem in which frequently used data values and instructions are duplicated for quick access.

call level interface

A library of function calls that support SQL statements and conform to the SQL Access Group Call Level Interface specification. These calls are typically used for dynamic access to data. ODBC is a call level interface.

call stack

An ordered list of functions that have been called but have not returned, with the currently executing function listed first. Each call is optionally shown with the arguments and types passed to it. During a debug session, you can view the functions that have been called but have not returned.

callback function

An application-defined function that a system or subsystem (Windows, for example) calls. Typically,

this happens when an event occurs or when windows or fonts are being enumerated. Examples of callback functions include window procedures, dialog-box procedures, and hook procedures. Callback functions are also used to process dynamic data exchange (DDE) transactions.

calling convention

A convention that determines the order in which arguments passed to functions are pushed on the stack (the calling sequence), whether the calling or called function removes the arguments from the stack, and the name-decorating convention the compiler uses to identify individual functions. See also C calling convention, calling sequence.

calling sequence

Determines the order in which parameters are pushed onto the stack during a function call and which code block is responsible for the stack pointer. Typically, the C compiler generates code that pushes parameters on the stack from right to left, beginning with the last parameter. See also calling convention.

caret

- 1 Or insertion point. A flashing line, block, or bitmap that marks the location at which inserted text will appear in a window's client area.
- 2 (^) When preceding a single uppercase letter, indicates a control character. For example, ^C is the same as CTRL+C.
- 3 (^) A regular expression used to indicate either the beginning of a line or, when used within brackets, to indicate an exception.

carriage return character

A control character that tells a computer or printer to return to the beginning of the current line. This character can have a different textual representation on different platforms, but it always has the ASCII value of 13. See also carriage return–linefeed (CR-LF) pair.

carriage return–linefeed (CR-LF) pair

The combination of a carriage return character (ASCII 13) and a linefeed character (ASCII 10), represented in C/C++ by the newline (\n) character.

casting

Explicit or implicit conversion of one data type to another.

catastrophic error

See fatal error.

catch block

Or catch handler. In C++, a block of exception-handling code preceded by the keyword **catch**. The code in the catch block is executed only if the code in the try block throws an exception of the type specified in the **catch** statement. See also C++ exception handling, throw expression.

CGI

A mechanism that allows a Web server to run a program or script on the server and send the

output to a Web browser. See also Internet Server Application Programming Interface (ISAPI).

Challenge Handshake Authentication Protocol

In ISDN, a type of signaling authentication that uses a pair of secret codes consisting of up to 16 characters. CHAP is shared by communications devices on both ends.

CHAP

In ISDN, a type of signaling authentication that uses a pair of secret codes consisting of up to 16 characters. CHAP is shared by communications devices on both ends.

character constant

A member of the source character set, the character set in which a program is written, surrounded by single quotation marks ('[.thsp]). Character constants are used to represent characters in the execution character set on the machine where the program executes.

character index

The number of characters from the beginning of an edit control.

character-mode application

An application that does not provide its own graphical user interface (GUI). The Win32 API provides consoles for managing input and output for character-mode applications. See also console application.

checksum

An error-detection scheme that involves creating a sum of the bits in a set of bytes of data and using that sum to later check for a change in the data. Checksums are commonly used in communications software to check for data transmission errors.

child control

A child window used in conjunction with another window (its parent) to carry out simple input and output (I/O) tasks.

child process

A process initiated by another process (the parent process). The child process can operate independently from the parent process. Further, the parent process can suspend or terminate without affecting the child process.

child window

A window that has the WS_CHILD or WS_CHILDWINDOW style and is confined to the client area of its parent window, which initiates and defines the child window. Typically, an application uses child windows to divide the client area of a parent window into functional areas. See also child control, sibling window.

chord

A closed figure bounded by the intersection of an ellipse and a line segment. In Windows, a chord is outlined by using the current pen and filled by using the current brush.

CL environment variable

An environment variable used to specify files and options for the compiler/linker so you do not have to specify them on the command line.

CL.EXE

Or CL. A driver program that controls the Microsoft C and C++ compilers and linker. The compilers produce Common Object File Format (COFF) object files. The linker produces executable (.EXE) files, dynamic-link libraries (DLLs), or static-link libraries.

class

A type that defines the interface of a particular kind of object. A class definition defines instance variables and methods, class variables and methods, and specifies the immediate superclass (or superclasses) and the interfaces that the class implements.

class declaration

In C++, the mechanism for declaring an aggregate data structure of type **class**. A class declaration provides a list of its members (such as functions, data, and other classes), specifies any friends of the class, and defines the level of visibility for all members.

class identifier

A universally unique identifier (UUID) that identifies a type of OLE object. Each type of OLE object (item) has its CLSID in the registry so that it can be loaded and programmed by other applications. For example, a spreadsheet may create worksheet items, chart items, and macrosheet items. Each of these item types has its own CLSID that uniquely identifies it to the system. See also registration entry file.

class library

A set of related C++ classes that can be used in an application, either as originally defined or as the source for other derived classes. The Microsoft Foundation Class Library included in Visual C++ is an example of a class library that defines a framework for integrating the user interface of an application for Windows with the rest of the application.

class method

In Java, any method that can be invoked using the name of a particular class. Since the declaration uses the keyword **static**, these are called static member functions in C++. Class methods, which are defined in class definitions, affect the class as a whole, not a particular instance of the class. Compare with instance method.

class scope

In C++, the degree of visibility afforded to a name (function or variable, for example) when it is declared within a class declaration. The name is accessible from outside the class by using the scope-resolution (::) operator. See also file scope, function scope, function-prototype scope, local scope.

class variable

In Java, a data item associated with a particular class as a whole, not with particular instances of the class. Class variables are defined in class definitions. Equivalent to static member variables in C++, since the declaration uses the keyword **static**. In C++, a class variable must be explicitly defined, external to the class declaration. See also instance variable.

ClassWizard file

A file that ClassWizard generates, containing information needed to edit existing classes or add new classes to a project. ClassWizard also uses the ClassWizard file to store information needed to create and edit message maps and dialog data maps, and to create prototype member functions.

ClassWizard files have a .CLW filename extension.

CLI

A library of function calls that support SQL statements and conform to the SQL Access Group Call Level Interface specification. These calls are typically used for dynamic access to data. ODBC is a call level interface.

client

An application or a process that requests a service from some other process, or from an in-process server. See also client/server.

client area

Or client rectangle. The portion of a window where the application displays output such as text or graphics.

client coordinates

An ordered pair (x,y) of numbers, relative to the origin (usually the upper-left corner of a window's client area), that designates a point in the client area. See also window rectangle.

client item

An object that provides an interface between an OLE item and the container application, and that is of a class derived from the MFC class **COleClientItem**. Client items are maintained by the container application and give the container application access to the presentation data and the native data. Client items also provide site(location) information to the server application for in-place activation. See also embedded item, linked item, server item.

client/server

- 1 The most commonly used model for distributed applications. Client applications request services from a server application. A server can have many clients at the same time, and a client can request data from multiple servers. An application can be both a client and a server. See also client.
- 2 In network architecture, a model for a local area network where clients initiate communication with the server, which carries out the requests in the form of replies. For example, the clients may be workstations communicating with a file server on which all of their data is stored. See also client.

clip path

A graphics object that an application can select into a device context. A clip path is always created by an application and it is used for clipping to one or more irregular shapes. For example, an application can use the lines and curves that form the outlines of characters in a string of text to define a clip path. See also clipping region.

clipboard

An area of storage, or buffer, where data objects or their references are placed when a user carries out a cut or copy operation.

clipboard format

The data format of a memory object on the clipboard. Applications can use the standard clipboard formats provided by Windows or register their own custom formats. A clipboard format is identified by a unique, unsigned integer value, called the format name.

clipboard owner

Or owner application. The application associated with the information on the clipboard. It is possible for there to be no clipboard owner. See also clipboard viewer.

clipboard viewer

A window that displays the contents of the clipboard. See also clipboard owner.

clipboard-viewer chain

A link between all of the running clipboard-viewer applications, enabling them to all receive the messages that Windows sends to the current clipboard viewer.

clipping

1 In Windows, the process of limiting output to a region or path within the client area in a window. For example, word processing and spreadsheet applications clip keyboard input to keep it from appearing in the margins of a page or spreadsheet.

2 In Open GL, eliminating the portion of a geometric primitive that is outside the half-space defined by a clipping plane.

clipping precision

A 16-bit value that defines how to clip characters that are partially outside the clipping region.

clipping region

In Windows, the portion of a window's client area where the system permits drawing.

CLSID

A universally unique identifier (UUID) that identifies a type of OLE object. Each type of OLE object (item) has its CLSID in the registry so that it can be loaded and programmed by other applications. For example, a spreadsheet may create worksheet items, chart items, and macrosheet items. Each of these item types has its own CLSID that uniquely identifies it to the system. See also registration entry file.

code page

A character set, which can include numbers, punctuation marks, and other glyphs. Different languages and locales may use different code pages. For example, code page 1252 is used for American English and most European languages. See also locale.

COFF

In 32-bit programming, a format for executable and object files that is portable across platforms. The Microsoft implementation of COFF is derived from the UNIX specification for COFF, but includes additional headers for compatibility with MS-DOS and 16-bit Windows. This Microsoft version is sometimes called the portable executable (PE) file format.

collection class

In object-oriented programming, a class that can hold and process groups of class objects or groups of standard types. A collection class is characterized by its shape (the way the objects are organized and stored) and by the types of its elements. MFC provides three basic collection shapes: lists, arrays, and maps (also known as dictionaries). See also collection object.

collection object

An object in a collection class.

color palette

An array containing the RGB values that identify the colors that can currently be displayed or drawn on the output device. Color palettes are used by devices that are capable of generating many colors but can only display or draw a subset of these at any given time. See also logical color palette.

COM

An open architecture for cross-platform development of client/server applications based on object-oriented technology as agreed upon by Digital Equipment Corporation and Microsoft Corporation. The Component Object Model defines an interface (similar to an abstract base class), **IUnknown**, from which all COM-compatible classes are derived.

combo-box control

In Windows, a child window that consists of a list box combined with either a static control or an edit control. The list-box portion of the control can either be displayed at all times or drop down when the user selects the drop-down arrow next to the control.

COMDAT record

A Common Object File Format (COFF) record that contains initialized common block data and makes packaged functions visible to the linker. See also packaged function.

command file

A text file that contains options and filenames you would otherwise type on the command line or specify using the CL or LINK environment variable. Since the command line is typically limited to 128 characters, a command file allows you to specify a large set of options or a very long file list to the compiler, linker, or resource compiler, for instance.

command identifier

Or command ID. In MFC, an identifier that associates a command message with the user-interface object (such as a menu item, toolbar button, or accelerator key) that generated the command. Typically, command IDs are named for the functionality of the user-interface object they are assigned to. For example, a Clear All item in the Edit menu might be assigned an ID such as ID_EDIT_CLEAR_ALL.

command line

A string of text typed at the command prompt, or executed from a command file, that specifies a task or tasks for the operating system or an application to perform.

command message

1 In Windows, a notification message from a user-interface object, such as a menu, toolbar button, or accelerator key. The framework processes command messages differently from other messages and such messages can be handled by a wider variety of object—documents, document templates, and the application object itself, in addition to windows and views.

2 In Media Control Interface (MCI), a symbolic constant that represents a unique command for an MCI device. Command messages have associated data structures that provide information a device requires to carry out a request.

comment delimiters

Characters used to denote text in a program that is not source code, thus telling the compiler to ignore it. C++ allows the traditional comment delimiters:

```
/* this is a comment */
```

as well as a single-line comment delimiter:

```
// everything else on this line is a comment
```

commit size

The amount of a resource that is allocated (or committed) for a particular use. For example, in the header of a COFF file, the Windows NT–specific field Heap Commit Size specifies the size of the local heap that the linker and loader should allocate for that file. See also reserve size.

common data record

A Common Object File Format (COFF) record that contains initialized common block data and makes packaged functions visible to the linker. See also packaged function.

common dialog box

A dialog box predefined in Windows that supports standard operations, such as the Open command on the File menu. An application displays a common dialog box by calling a single function rather than by supplying a dialog box procedure and using a resource file containing a dialog box template.

Common Gateway Interface

A mechanism that allows a Web server to run a program or script on the server and send the output to a Web browser. See also Internet Server Application Programming Interface (ISAPI).

Common Object File Format

In 32-bit programming, a format for executable and object files that is portable across platforms. The Microsoft implementation of COFF is derived from the UNIX specification for COFF, but includes additional headers for compatibility with MS-DOS and 16-bit Windows. This Microsoft version is sometimes called the portable executable (PE) file format.

compact executable file

An executable binary (program) file whose code is limited to a single 64-kilobyte segment. Compact executable files usually have a .COM filename extension and are often used for utility programs and short routines. See also executable file.

compilation

The translation of source code into object code.

compilation unit

The smallest unit of code that can be independently compiled, usually a source code file. See also translation unit.

compile time

The point at which a program is being compiled, or the amount of time required to perform a compilation of a program.

compile-time error

A syntactic or semantic error that prevents a program from being compiled.

compiled resource file

Or binary resource file. A binary file that contains a Windows-based application's resource data and is created by the resource compiler from the resource-definition (.RC) file. Compiled resource files usually have a .RES filename extension. See also Macintosh binary resource file, resource compiler.

compiler

A program that translates source code, such as C++ or Pascal, into directly executable machine code.

compiler/linker driver

Or CL. A driver program that controls the Microsoft C and C++ compilers and linker. The compilers produce Common Object File Format (COFF) object files. The linker produces executable (.EXE) files, dynamic-link libraries (DLLs), or static-link libraries.

complete object

An instance of a derived class from which no other classes are derived. A complete object is an object that is not a subobject representing a base class.

complex type

See aggregate type.

Component Object Model

An open architecture for cross-platform development of client/server applications based on object-oriented technology as agreed upon by Digital Equipment Corporation and Microsoft Corporation. The Component Object Model defines an interface (similar to an abstract base class), **IUnknown**, from which all COM-compatible classes are derived.

compositing

The process of superimposing one image on another to create a single image.

compound document

Or container document. A document within a container application that contains data of

different formats, such as sound clips, spreadsheets, text, and bitmaps. Each piece of integrated data (or compound-document object) can exist within the compound document as a linked item or an embedded item.

compound file

The OLE implementation of the structured-storage model, which specifies how data is saved to and retrieved from storage. Conceptually, a compound file is a number of individual files (or stream objects) multiplexed into one physical file (or storage object) that still allows access to each individual file.

Compressed SLIP

SLIP with data compression for a more efficient connection. See also Serial Line Internet Protocol (SLIP).

conditional expression

Or Boolean expression, logical expression. An expression that yields a Boolean value (true or false). Such expressions can involve comparisons, using relational operators such as the less-than (<) and greater-than (>) operators, and logical combination of Boolean expressions, using Boolean operators such as bitwise AND (&) and logical OR (||).

connection string

Or connect string. In ODBC, a string expression used to open an external database.

console

An interface that provides input and output to character-mode applications. This processor-independent mechanism makes it easy to port existing character-mode applications or to create new character-mode tools and applications.

console application

- 1 A character-mode application that uses a console window for its input and output. If necessary, the operating system will create a new console window, which exists until the application terminates.
- 2 More generally, a program that runs from the operating system's command line, in character-mode, rather than from a graphical user interface.

constant

An object or variable that is not modifiable. In C++, the keyword **const** can be used to define constant values. See also manifest constant.

constant expression

An expression that is evaluated at compile time instead of run time. The constant expression can be used in any place that a constant can be used, but it must evaluate to a constant that is in the range of representable values for that type.

constructor

In C++, a special initialization function that is called automatically whenever an instance of a class is declared. This function prevents errors that result from the use of uninitialized objects. The constructor must have the same name as the class itself and must not return a value. See also copy

constructor, default constructor, destructor.

container application

Or OLE container. An application that can incorporate embedded or linked items into its own documents. The documents managed by a container application are able to store and display OLE Visual Editing items as well as data created by the application itself. A container application allows users to insert new items or edit existing items. See also server application.

context identifier (ID)

Or context reference. A unique number or string that corresponds to a particular object in the application—for example, to a menu command, form, control, or screen region. Context IDs are used to create links between the application and the corresponding Help topics.

context number

The number used to identify a Windows Help topic. If context numbers are not explicitly assigned to topics, the Help compiler generates default values by converting topic strings into context numbers. The [MAP] section of a Help project (.HPJ) file associates a context string and a context number. See also context string.

context string

A unique character string formatted as hidden text in a rich-text format (.RTF) file. Context strings link hot spots to target topics. The [MAP] section of a Help project (.HPJ) file associates a context string and a context number. See also context number.

control bar

A window that can contain buttons, edit boxes, check boxes, or other kinds of Windows controls. A control bar is usually aligned with the top or bottom of a frame window and provides quick, one-step command actions. Control bars include toolbars, status bars, and dialog bars.

control identifier (ID)

A 16-bit value that an application uses to uniquely identify a child control. This ID is used in notification messages to the parent window when events, such as input from the user, occur in the control.

conversion function

1 In C++, a special member function that makes an explicit conversion from a given class type to another data type by using the **operator type-name()** syntax. Conversion functions are often called cast operators because they are the functions called when a cast operator is used.

2 More generally, any function that converts one data type or format to another data type or format.

coordinate space

A planar space based on the Cartesian coordinate system.

coordinated universal time

A global time standard equivalent to Greenwich mean time (GMT).

copy constructor

In C++, a constructor with one parameter, whose type is a reference to another instance of the class. If a class is declared without a copy constructor, the compiler will generate one automatically. The copy constructor is used when an instance is created from another instance (for example, an assignment from one instance of the class to another).

CRC

An error-detecting method that uses a polynomial code. The method is sometimes referred to as the polynomial code.

critical section

A segment of code which is not reentrant; that is, it does not support concurrent access by multiple threads. Often, a critical section is used to protect shared resources

cross compilation

A compilation of source code that takes place on one hardware platform but generates object code for another. For example, object code for the Power Macintosh can be compiled on an Intel-based Windows platform. See also compilation, object code.

CSLIP

SLIP with data compression for a more efficient connection. See also Serial Line Internet Protocol (SLIP).

cursor resource file

A file that contains an image that defines the shape of a cursor on the screen. Cursor resource files usually have a .CUR filename extension.

custom control

A special-format dynamic-link library (DLL) or object file that adds features and functionality to a Windows-based application user interface. A custom control can be a variation on an existing Windows dialog-box control (for example, a text box suitable for use with a pen and digitizing tablet) or an entirely new category of control. See also ActiveX control.

custom resource

Or application-defined resource. A resource that a developer creates and adds to an application's executable file that contains data required by the application. See also standard resource.

cyclic redundancy check

An error-detecting method that uses a polynomial code. The method is sometimes referred to as the polynomial code.

Glossary D

DAG

In programming, an abstract data type often used to represent arbitrary relationships between objects. The graph consists of nodes (data objects) connected by paths that have direction; there are no cycles in the graph. In object-oriented program design, DAGs are useful for depicting inheritance relationships among classes.

DAO

A high-level set of objects that insulates developers from the physical details of reading and writing records. In a database application, for example, these objects include databases, table definitions, query definitions, fields, indexes, etc.

DAO record field exchange

The mechanism by which the MFC DAO classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), bulk record field exchange (Bulk RFX), and dialog data exchange (DDX).

Data Access Objects

A high-level set of objects that insulates developers from the physical details of reading and writing records. In a database application, for example, these objects include databases, table definitions, query definitions, fields, indexes, etc.

data binding

A notification mechanism that links control properties through the container to a data source, such as a database field.

data declaration

A statement within a program that specifies the characteristics of a variable. Most programming languages allow (or require) specification of a variable's name and data type and possibly its initial value as well. Array declarations usually require a size specification in addition to the name and type, and record declarations must specify the elements of the record. See also data type, type declaration.

data definition language

Or database design language, data design language. A language, usually a part of a database management system, that defines all attributes and properties of a database, especially record layouts, field definitions, key fields (and, sometimes, keying methodology), file locations, and storage strategy.

data field

- 1 A well-defined portion of a data record, such as a column in a database table.
- 2 The physical representation of a unit of data.

data file

A file consisting of data—text, numbers, or graphics, for example. Such a file is distinct from a

program file of executable instructions.

data fork

The portion of an Apple Macintosh file that contains user-supplied information. See also resource fork.

data handle

In Microsoft Windows, a 32-bit value used to provide access to data in a dynamic data exchange (DDE) object.

data map

In MFC, a mechanism that automates the process of gathering values from a dialog box by providing functions to initialize the controls in the dialog box with the proper values, retrieve the data, and validate the data.

data member

A data object defined as part of a class. See also member function.

data segment

The portion of memory or auxiliary storage that contains the data needed by an application or a portion of an application. Usually, the data segment is readable and writable.

data source

In ODBC, a specific set of data, the information required to access that data, and the location of the data source, which can be described using a data source name. From a program's point of view, the data source includes the data, the DBMS, the network (if any), and ODBC.

data source name

The name of a data source that applications use to request a connection to the data source. For example, a data source name can be registered with ODBC through the ODBC Administrator program.

data structure

An organizational scheme, such as a record or an array, applied to data so that it can be interpreted and so that specific operations can be performed on that data.

data symbol

The name that identifies the memory location of a global or static data object. The concept of data symbol includes all data objects except local (stack-allocated) or dynamically allocated data.

data type

In programming, a definition of a set of data that specifies the possible range of values of the set, the operations that can be performed on the values, and the way in which the values are stored in memory. See also aggregate type, scalar type.

data-access application

An application used to access data, typically in a database. See also database application.

database application

An application that manages files consisting of a number of records (or tables), each of which is constructed of fields (columns) of a particular type, together with a collection of operations that facilitate searching, sorting, recombination, and similar activities. See also data-access application, relational database.

database form

A structured window, box, or other self-contained presentation element built into a database application that allows the user to perform a variety of data-access tasks, including data entry, read-only examination of data, and data updates. The form acts as a visual filter for the underlying data it is presenting, generally offering the advantages of better data organization and greater ease of viewing.

database management system

A layer of software between the physical database and the user. The DBMS manages all requests for database action (for example, queries or updates) from the user. Thus, the user is spared the necessity of keeping track of the physical details of file locations and formats, indexing schemes, and so on. In addition, a DBMS may permit centralized control of security and data-integrity requirements.

database schema

A description of the current structure of tables and views in a database. The schema describes the columns in each table and the data type of each column.

datagram socket

A connectionless socket that provides a bidirectional flow of data. Datagram data may arrive out of order and possibly duplicated, but record boundaries in the data are preserved, as long as the records are smaller than the receiver's internal size limit. An example of a datagram socket is an application that keeps system clocks on the network synchronized. See also stream socket, transport protocol.

DBCS

A character set that can be used to represent Far Eastern languages that use ideographic characters. Like a multibyte character set (MBCS), a DBCS contains both single- and double-byte characters. DBCS characters are addressed using two bytes. The DBCS single-byte characters conform to the 8-bit national standards for each country and correspond closely to the ASCII character set. See also lead byte, trail byte, Unicode.

DBMS

A layer of software between the physical database and the user. The DBMS manages all requests for database action (for example, queries or updates) from the user. Thus, the user is spared the necessity of keeping track of the physical details of file locations and formats, indexing schemes, and so on. In addition, a DBMS may permit centralized control of security and data-integrity requirements.

DC

A data structure defining the graphic objects, their associated attributes, and the graphic modes affecting output on a device. See also graphic object, metafile.

DDE

A form of interprocess communications that uses shared memory to exchange data between

applications. DDE can be used for one-time data transfers and for ongoing exchanges by applications that send updates to one another as new data becomes available. See also client/server, OLE Automation.

DDL

Or database design language, data design language. A language, usually a part of a database management system, that defines all attributes and properties of a database, especially record layouts, field definitions, key fields (and, sometimes, keying methodology), file locations, and storage strategy.

DDV

In MFC, a method for checking data as it is transferred from the controls in a dialog box. DDV is an easy way to validate data entry in a dialog box. See also dialog data exchange (DDX).

DDX

In MFC, a method for transferring data between the controls of a dialog box and their associated variables. DDX is an easy way to initialize the controls in a dialog box and to gather data input by the user. See also dialog data validation (DDV).

dead key

On some non-English keyboards, a key that is used with another key to create an accented character. A dead key, when pressed, produces no visible character but indicates that the diacritic it represents is to be combined with the character produced by the next letter key pressed.

deadlock

A state in which every process in a set of processes is waiting for an event or resource that only another process in the set can provide. For example, in data communications, a deadlock can occur when both the sending and receiving sockets are waiting on each other, or for a common resource.

debug memory allocator

A memory allocation function that allows you to validate and track the memory allocated for your objects on the global heap. For example, the Debug version of the Microsoft Foundation Class Library has memory allocation functions.

debug monitor

Or remote monitor. A small program that resides on the target machine and controls the execution of the remote program and communication with the Microsoft Developer Studio debugger. See also remote debugging.

debug terminal

A dumb terminal or computer other than the developer's primary machine, to which debugging output is sent.

debug version

- 1 A version of a program built with symbolic debugging information. See also release version.
- 2 Or debug library. A library version (for example, of the Microsoft Foundation Class Library) that includes diagnostic aids and performs various integrity checks to aid in debugging a program. See

also release version.

debugger

A program designed to help find errors in another program by allowing the programmer to step through the program, examine data, and check conditions. There are two basic types of debuggers. Machine-level debuggers display the actual machine instructions (disassembled into assembly language) and allow the programmer to look at registers and memory locations. Source-level debuggers let programmers look at the original source code, examine variables and data structures by name, and so on.

debugging event

An incident in the process being debugged that causes the kernel to notify the debugger. Debugging events include creating a process, creating a thread, loading a dynamic-link library (DLL), unloading a DLL, sending an output string, and generating an exception.

debugging information

Symbolic information used by a debugger, especially information in the Microsoft Symbolic Debugging Information format that is used by the Microsoft CodeView debugger. Also, any data (information) generated by the debugging process.

declaration

A statement that binds an identifier to the information that relates to it. For example, to declare a constant means to bind the name of the constant with its value. To declare a variable means to bind the variable's name with a location in memory and with the information about the variable's data type. Declaration usually occurs within the source code of a program; the actual binding can take place at compile time or run time. Declaration can be performed explicitly (by specifying the identifier and relevant information in a declare statement) or implicitly (by using the undeclared identifier in a statement), depending on the language being used.

declaration statement

A statement used to declare the names and data types of constants, variables, user-defined data types, and procedures. In C++, the declaration statement can also be an executable statement. For example:

```
int i = fCount( );
```

declares an integer variable i and assigns it the value returned from the call to `fCount()`.

declarator

The part of a declaration that names an object, type, or function. In C/C++, declarators appear in a declaration as one or more names separated by commas; each name can have an associated initializer.

decorated name

Or type-safe name, mangled name. In C++, a compiler-generated string that contains an undecorated (literal) name followed by a string of characters that the compiler and linker use to retain type information. Name decoration is necessary to resolve ambiguities that arise in C++ from having more than one function, for example, with the same name.

decrement (--) operator

An operator that specifies that its integral or floating-point operand be decreased by the integer value 1. A decremented pointer points to the previous object. In the prefix form (`--i`), the decrement takes place before the value is used in expression evaluation; in the postfix form (`i--`), the decrement takes place after the value is used in expression evaluation. See also increment (`++`) operator.

default argument

In a C++ function definition, an argument with an assigned value that specifies the value an argument should assume if none is supplied in the function call.

default constructor

In C++, a constructor that either accepts no arguments or for which all arguments have a default value. The default constructor can be defined by the user or generated by the compiler. See also constructor, copy constructor.

default message processing

Message processing carried out by a default window procedure on any window messages for which there is not an explicit procedure defined in the application. In applications based on the Microsoft Foundation Class Library, **DefWindowProc** is the default window procedure.

default value

A value that the system or software assumes, unless the user makes an explicit choice.

default window procedure

A system-defined function that defines certain fundamental behavior shared by all windows. A default window procedure provides default processing for nonclient-area messages, system commands, system keystrokes, and other messages that the application-defined window procedure does not specifically handle. See also window procedure.

definition

A construct that initializes and allocates storage for a variable, function, or class. For a function, the definition specifies the name, formal parameters, body, and return type; for a class, the definition specifies its data members and functions. See also declaration.

delayed rendering

Formatting data when it is requested, rather than when it first becomes available. For example, when placing a Clipboard format on the Clipboard, a window can delay transferring the data to that format until the data is needed. This is useful if the application supports several Clipboard formats, some or all of which are time-consuming to render.

delimiter

A special character that sets off, or separates, individual items in a program or in a set of data. Programming languages typically delimit such variable-length elements as comments, strings, and program blocks. Databases use two forms of delimiters: field delimiters and record delimiters. Characters used as delimiters include commas, semicolons, tabs, carriage returns, and colons.

derived class

In object-oriented programming, a class that is created from another class, called the base

class. A derived class inherits all the features of its base class. See also inheritance, polymorphism.

descendant

1 In object-oriented programming, a class that is derived from another class. See also base class, derived class, inheritance.

2 Generally, a process or task that is called by another process or task and whose characteristics and behavior are determined, at least in part, by the originating process or task. See also child process, child window.

descendant window

A child window that is derived from a parent window. A child window has only one parent window, but a parent can have any number of child windows. Each child window, in turn, can have child windows. In this chain of windows, each child window is called a descendant window of the original parent window.

description block

In a makefile, a dependency line that specifies a block of commands to run if the listed targets and their dependents are out of date.

deserialization

Re-creating an object by reading its state from persistent storage (usually a file) and reconstructing it in memory. See also serialization.

Desktop Management Interface

A protocol-independent, multiplatform interface for workstation and network components.

desktop window

A system-defined window that paints the background of the screen and serves as the base for all windows displayed by all Windows-based applications.

destructor

In C++, a member function that is automatically called when a class object is destroyed (either by being explicitly deallocated or by going out of scope). Its purpose is to perform any cleanup work necessary before an object is destroyed. The destructor's name is the class name with a tilde (~) as a prefix. See also constructor.

development environment

In the Microsoft Developer Studio, an integrated set of Windows-based tools for completing, testing, and refining an application. Microsoft Developer Studio includes a text editor, resource editors, project build facilities, an optimizing compiler, an incremental linker, a source code browse window, and an integrated debugger.

device context

A data structure defining the graphic objects, their associated attributes, and the graphic modes affecting output on a device. See also graphic object, metafile.

device driver

A low-level software component that permits device-independent software applications to communicate with a device such as a mouse, keyboard, monitor, or printer.

device-independent

A characteristic of software and files that generate the same output regardless of the hardware involved. A device-independent program could, for example, issue the same command to draw a rectangle regardless of whether the output device was a printer, a plotter, or a screen display.

device-independent bitmap file

A file containing an array of bits combined with several structures that specify the width and height of the bitmapped image (in pixels), the color format of the device where the image was created, and the resolution of the device used to create that image. The DIB file format ensures that bitmap graphics created in one application can be loaded and displayed in another application exactly the way they appear in the originating application. See also bitmap, bitmap file.

device-mode setting

A user-selected setting that contains information used by the relevant device driver. For example, device-mode settings for a laser printer setup could specify whether printing is to be in landscape or portrait mode, the size of paper used, and the quality of printing.

DFX

The mechanism by which the MFC DAO classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), bulk record field exchange (Bulk RFX), and dialog data exchange (DDX).

diagnostic services

Services that facilitate program debugging. Diagnostic services include functions and macros to resolve assertions, errors, and exceptions; trace memory allocations; and resolve leaks; and report debug messages to the user—all during run time. In Microsoft Visual C++, most of these services require the debug version of the C run-time library.

dial on demand

In ISDN, a dial-up router function that activates a remote link only when data needs to be sent.

dialog bar

A control bar that contains standard Windows controls. A dialog bar has dialog-box characteristics in that it contains controls and supports tabbing between them, and it uses a dialog template to represent the bar. Dialog bars can be aligned to the top, bottom, left, or right side of a frame window. See also status bar.

dialog box

In Windows, a child window used to retrieve user input. A dialog box usually contains one or more controls, such as buttons, list boxes, combo boxes, and edit boxes, with which the user enters text, chooses options, or directs the action of the command.

dialog data exchange

In MFC, a method for transferring data between the controls of a dialog box and their associated variables. DDX is an easy way to initialize the controls in a dialog box and to gather data input by

the user. See also dialog data validation (DDV).

dialog data validation

In MFC, a method for checking data as it is transferred from the controls in a dialog box. DDV is an easy way to validate data entry in a dialog box. See also dialog data exchange (DDX).

dialog editor

A resource editor that allows you to place and arrange controls in a dialog-box template and to test the dialog box. The editor displays the dialog box exactly as the user will see it. While using the dialog editor, you can define message handlers and manage data gathering and validation with the ClassWizard.

dialog file

A file that contains dialog-box source code. Note that a dialog file is not required for a Visual C++ project because Visual C++ keeps this code in the resource-definition file.

dialog template

A template used by Windows to create a dialog window and display it. The template specifies the characteristics of the dialog box, including its overall size, initial location, and style, and the types and positions of its controls. A dialog template is usually stored as a resource, but templates can also be stored directly in memory. See also dialog file, dialog editor, resource-definition file.

dialog unit

A unit of horizontal or vertical distance within a dialog box. A horizontal DLU is the average width of the current dialog-box font divided by 4. A vertical DLU is the average height of the current dialog-box font divided by 8.

digital signature

An electronic identifier used for security. It verifies that the document originated from the individual whose signature is attached to it and that it has not been altered since it was signed. Usually accomplished using some form of encryption.

dimension

The zero-based number or numbers inside brackets in an array declaration that define the array's size. For example, `char p[10]` declares a one-dimensional character array that has ten elements. To declare arrays of two or more dimensions, place each dimension in its own set of brackets, as in `char x[10][20]`.

dimmed

Or disabled, grayed, unavailable. The state and visual appearance of controls or menu items whose functionality is not presently available to a user.

direct mode

A file-access mode that saves changes to the document as they are made. See also transacted mode.

directed acyclic graph

In programming, an abstract data type often used to represent arbitrary relationships between objects. The graph consists of nodes (data objects) connected by paths that have direction; there are no cycles in the graph. In object-oriented program design, DAGs are useful for depicting inheritance relationships among classes.

dirty

Indicates that a file, object, or data item has been changed since the last time it was saved. Usually describes a flag or bit that is set to 1 when data changes and back to 0 when the data is saved to disk.

disjoint figure

A figure that consists of a set of points that are not logically sequential or physically adjacent to each other.

disk drive

A physical device that reads from or writes to disks. Disk drives are referenced by letters, typically A: for the first floppy-disk drive, B: for the second floppy-disk drive, C: for the first fixed-disk drive, D: for the second fixed-disk drive, and so on. Each drive (whether physical or logical) on an operating system is assigned a unique letter to distinguish it from other drives.

dispatch identifier (ID)

A 32-bit attribute value for identifying methods and properties in OLE Automation. All of the accessor functions for a single property have the same dispatch ID. The low-order 16 bits of the dispatch ID contain the distance from the top of the dispatch map; the high-order 16 bits contain the distance from the most derived class.

dispatch interface

In OLE Automation, the external programming interface of some grouping of functionality exposed by the automation server. For example, a dispatch interface might expose an application's mouse clicking and text data entry functions. See also type library (.TLB) file.

dispatch map

In MFC, a set of macros that expands into the declarations and calls needed to expose methods and properties for OLE Automation. The dispatch map designates the internal and external names of object functions and properties, as well as the data types of the function arguments and properties.

display device context

A device context, created by Windows, that an application uses to paint and draw on a video display. Windows prepares a display device context for each window, or the screen, setting the drawing objects, colors, and modes for the device.

Distributed Management Environment

A strategy developed to manage distributed heterogeneous networks that conform to DCE.

dithering

A technique for increasing the perceived range of colors in an image at the cost of spatial resolution. Adjacent pixels are assigned differing color values; when viewed from a distance, these colors seem to blend into a single intermediate color. The technique is similar to the half-toning used in black-and-white publications to achieve shades of gray.

DLU

A unit of horizontal or vertical distance within a dialog box. A horizontal DLU is the average width of the current dialog-box font divided by 4. A vertical DLU is the average height of the current dialog-box font divided by 8.

DME

A strategy developed to manage distributed heterogeneous networks that conform to DCE.

DMI

A protocol-independent, multiplatform interface for workstation and network components.

dockable toolbar

A toolbar that can be attached, or docked, to any side of its parent window, or floated in its own mini-frame window. See also docked toolbar, floating toolbar.

docked toolbar

A toolbar that is attached, or docked, to any side of its parent window. See also dockable toolbar, floating toolbar.

document

- 1 (noun) Any self-contained piece of work created with an application program and, if saved on disk, given a unique filename by which it can be retrieved.
- 2 (verb) To explain or annotate something, such as comments within a program or a description of a problem in a bug report.

document item

An object of a class derived from the **CDocItem** class, encapsulating some component of a document's data. Document items are used to represent OLE items in both client and server documents. See also client item, server item.

document object

An object that defines, stores, and manages an application's data. When the user opens an existing or new document, the application framework creates a document object to manage the data stored in the document.

document template

In MFC, a template used for the creation of documents, views, and frame windows. A single application object manages one or more document templates, each of which is used to create and manage one or more documents (depending on whether the application is SDI or MDI). Applications that support more than one type of document, such as spreadsheets as well as text, have multiple document template objects.

document window

In windowing environments, an on-screen window (enclosed work area) in which the user can create, view, or work on a document.

document/view architecture

A design methodology that focuses on what the user sees and needs rather than on the application or what the application requires. This design is implemented by a set of classes that manage, store and present application-specific data. These classes can manipulate disk-based data files (document objects), display a document's data (view objects), and automatically use a particular type of window (window objects).

domain name

- 1 In general networking, a logical grouping of machines for network administration purposes.
- 2 In TCP/IP, the unique name that identifies an Internet or network site. Domain names always have two or more parts, separated by dots (for example, **microsoft.com** or **www.microsoft.com**). The part on the left is the most specific, and the part on the right is the most general. A given machine may have more than one domain name, but a given domain name points to only one machine.

dots per inch

A measure of screen and printer resolution that is expressed as the number of dots per horizontal or vertical inch that a device can print or display .

double-byte character set

A character set that can be used to represent Far Eastern languages that use ideographic characters. Like a multibyte character set (MBCS), a DBCS contains both single- and double-byte characters. DBCS characters are addressed using two bytes. The DBCS single-byte characters conform to the 8-bit national standards for each country and correspond closely to the ASCII character set. See also lead byte, trail byte, Unicode.

doubleword

Or DWORD. A unit of data consisting of 4 contiguous bytes that are processed as a single unit by a computer's microprocessor.

DPI

A measure of screen and printer resolution that is expressed as the number of dots per horizontal or vertical inch that a device can print or display .

drag-and-drop

A technique for moving or copying data between applications, between windows within an application, or within a single window in an application. The user selects the data to be transferred and drags the data to the desired destination.

driver

- 1 A hardware device that controls or regulates another device. A line driver, for example, boosts signals transmitted over a communications line, and a bus driver amplifies and regulates signals transmitted over a bus (data pathway).
- 2 A program that controls a device such as a printer or a mouse. See also device driver.

drop source

In a drag-and-drop operation, the window from which the user selects data for transfer. See also drop target.

drop target

In a drag-and-drop operation, the destination window where the user drops the data being transferred. See also drop source.

drop-down combo box

A combo box that contains a drop-down list and a selection field that the user can edit.

drop-down list

A list in a combo box that displays the current setting, but can be opened to display a list of choices. The user can select an item from the list to update the current setting.

drop-down menu

A menu that is displayed when the user selects a particular entry from a menu bar. See also pop-up menu.

DSN

The name of a data source that applications use to request a connection to the data source. For example, a data source name can be registered with ODBC through the ODBC Administrator program.

dual interface

An interface that supports both IDispatch and VTBL binding.

DUMPBIN

The Microsoft COFF Binary File Dumper (DUMPBIN.EXE). DUMPBIN displays information about 32-bit Common Object File Format (COFF) binary files. DUMPBIN can be used to examine COFF object files, standard libraries of COFF objects, executable files, and dynamic-link libraries (DLLs). DUMPBIN is a 32-bit tool that runs only from a command prompt.

dynamic creation

The process of creating an object of a specific class at run time. Do not confuse this dynamic creation of an object with the creation of a dynamic object, using the C++ **new** operator. Dynamic creation is not supported directly by the C++ language. Objects derived from the MFC class **CObject** can have this functionality.

dynamic data exchange

A form of interprocess communications that uses shared memory to exchange data between applications. DDE can be used for one-time data transfers and for ongoing exchanges by applications that send updates to one another as new data becomes available. See also client/server, OLE Automation.

dynamic link

A program link, to a library or to an object, that is established when the program is loaded into memory (load-time dynamic linking) or while it is running (run-time dynamic linking). See also dynamic-link library (.DLL) file, static link.

dynamic priority

A thread priority value used by the scheduler in making scheduling decisions. The value for each thread can never be lower than the thread's base priority, but it can be raised and then lowered to enhance responsiveness to input or other significant events.

dynamic splitter window

A split-window style in which additional panes are created and destroyed as the user splits and unsplit views. Microsoft Excel and Microsoft Word are examples of applications that offer the dynamic splitter style. See also static splitter window.

dynamic-link library file

A file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. In Win32, the operating system maps the dynamic-link libraries (DLLs) into the address space of a process when the process is starting up or while it is running. The process then executes functions in the DLL. Dynamic-link library files usually have a .DLL filename extension.

dynaset

A recordset (or set of records) with dynamic properties that is the result of a query on a database document. A dynaset can be used to add, change, and delete records from the underlying database table or tables. See also snapshot.

Glossary E

edit buffer

In MFC database classes, a buffer that contains the current record during an update. Update operations may use this buffer to manage changes to the data source.

edit control

Or edit box, text box. A rectangular control window that a user can use to type and edit text. See also static control.

EDITBIN

Or COFF Binary File Editor. The Microsoft 32-Bit binary file editor for modifying 32-bit Common Object File Format (COFF) binary files. Use EDITBIN to modify object files, executable files, and dynamic-link libraries (DLLs). You can also use EDITBIN to convert the format of an Object Model Format (OMF) input file to COFF.

embedded item

Or embedded object. A type of compound-document item in which all the information needed to manage the item is stored in the container document, but which is created and edited by a server application. Embedded items can be edited or activated in-place. See also linked item.

encapsulation

In object-oriented programming, the process of hiding the internal workings of a class to support or enforce abstraction. A class's interface, which is public, describes what a class can do, while the implementation, which is private or protected, describes how it works.

encryption

The transformation of data into a form unintelligible to anyone without a secret decryption key and algorithm. Its purpose is to ensure privacy by keeping the information hidden from anyone for whom it is not intended.

end of file

A value returned by an I/O routine when the end of a file (or, in some cases, an error) is encountered. The iostream library function **eof** returns TRUE on the end-of-file condition. When a file is opened in text mode, a logical end of file occurs whenever a CTRL+Z character is encountered.

entry point

A starting address for a function, executable file, or dynamic-link library.

environment variable

A symbolic variable that represents an element of the user's operating system environment, such as a path, a directory name, or a configuration string. For example, the environment variable PATH represents the directories to search for executable files.

EOF

A value returned by an I/O routine when the end of a file (or, in some cases, an error) is encountered.

The iostream library function eof returns TRUE on the end-of-file condition. When a file is opened in text mode, a logical end of file occurs whenever a CTRL+Z character is encountered.

epilog code

See prolog/epilog code sequence.

error message

A message from the system or a program advising the user of a problem that requires human intervention in order to be solved.

escape sequence

In C/C++, a character combination consisting of a backslash (\) followed by a letter or by a combination of digits. An escape sequence is regarded as a single character and is therefore valid as a character constant. Escape sequences are typically used to provide literal representations of nonprinting characters, such as the newline character (\n) and characters that have special meanings in the C/C++ language, such as the double quotation mark (\").

event

- 1 In OLE, a notification message sent from one object to another (e.g. from a control to its container) in response to a state change or a user action.
- 2 More generally, any action or occurrence, often generated by the user, to which a program might respond. Typical events include keystrokes, mouse movements, and button clicks.

event object

A synchronization object that allows one thread to notify another that an event has occurred. Event objects are useful when a thread needs to know when to perform its task. For example, a thread that copies data to a data archive would need to be notified when new data is available. By using an event object to notify the copy thread when new data is available, the thread can perform its task as soon as possible.

exception

An abnormal condition or error that occurs during the execution of a program and that requires the execution of software outside the normal flow of control. Examples of exceptions are running out of memory, resource allocation errors, and failure to find files. See also C++ exception handling, structured exception handling (SEH).

exception handler

A block of code that reacts to a specific type of exception. If the exception is for an error from which the program can recover, the program can resume executing after the exception handler has executed. In this case, execution will resume where the exception was handled, not at the place where it was generated.

executable file

A program file created from one or more source code files translated into machine code and linked together. The MS-DOS, Windows, and Windows NT operating systems use the .EXE filename extension to indicate that the file is a runnable program.

execution character set

The character set on the machine where the program executes. For Microsoft C and C++, the execution set is the standard ASCII character set. See also source character set.

explicit initializer

An initial value that is explicitly stated when a program variable is declared. In C++, a single initializer can be supplied with the simple-assignment (=) operator, as follows:

```
int nCount = 0;
```

or an initializer list can be supplied, enclosed in parentheses:

```
CString strFileName(FILE.DAT);  
CRect *pRect = new CRect(10, 15, 24, 97);
```

exported function

One called by other executing entities, such as DLLs or executable files. Typically, a DLL will need to export functions to allow its clients to control it. Functions can be exported by using either a module-definition (.DEF) file or the __declspec (dllexport) storage-class modifier, a Microsoft-specific extension to the C language. This modifier ensures that other applications and dynamic-link libraries will be able to obtain, dynamically at runtime, the address of the variable or function to which it applies. See also imported function.

exports file

A file that contains information about exported functions and data items. The Microsoft 32-Bit Library Manager tool (LIB.EXE) generates the exports file from the module-definition (.DEF) file. The linker uses the exports file to build the dynamic-link library (.DLL) file. Exports files have a .EXP filename extension.

expression

A sequence of tokens that can be evaluated.

expression statement

An expression followed by a semicolon (;). All expressions in an expression statement are evaluated and all side effects are completed before the next statement is executed. The most common expression statements are assignments and function calls.

external linkage

Specifies the way the names of objects and functions are shared between translation units. With external linkage, names can refer to program elements in any translation unit in the program—the program element is shared among the translation units and the same name in another translation unit is guaranteed to refer to the same object or class. Names with external linkage are sometimes termed global. The keyword **extern** before a name declaration ensures external linkage. See also internal linkage.

external name

- 1 In OLE Automation, an identifier that a class exposes to other applications. Automation clients use the external name to request an object of this class from an automation server.
- 2 In C/C++, an identifier declared with global scope or declared using the **extern** storage class.

extraction (>>) operator

Or get-from operator, input operator. In C++, the right-shift operator, overloaded (in the iostream library) to accept input. The left operand must be the predefined input stream **cin** and the right operand must be a variable. See also insertion (<<) operator.

Glossary F

F1 Help

Context-sensitive Windows Help that the user obtains by pressing the F1 key. F1 Help opens Help on a topic associated with the currently selected item in the application. MFC supplies F1 Help for windows, dialog boxes, message boxes, menus, and toolbar buttons.

FAQ

Documents that list and answer the most common questions on a particular subject.

far pointer

Or long pointer. In 16-bit programming, a 32-bit pointer, which is a pointer that can point anywhere in memory because it specifies both the segment and the offset for a memory location. In 32-bit programming, all pointers are 32 bits wide and there is no need to distinguish between near and far pointers.

FAT

A data member of a recordset object that corresponds to a particular column (or field) in the query that the recordset represents. Collectively, these field data members make up a buffer that holds the fields of the current record. When the user scrolls to a new record, the database framework replaces the previous record's values with the values of the new current record.

fatal error

Or unrecoverable error. An error that causes the system or a program to fail abruptly with no hope of recovery. An example of a fatal error is an uncaught exception that cannot be handled.

favorite

A reference in Internet Explorer to a page to which the user may want to return. Corresponds to bookmark in other browsers.

field data member

A data member of a recordset object that corresponds to a particular column (or field) in the query that the recordset represents. Collectively, these field data members make up a buffer that holds the fields of the current record. When the user scrolls to a new record, the database framework replaces the previous record's values with the values of the new current record.

file allocation table file system

The file system that MS-DOS uses to store information on disks. The file allocation table, which the operating system creates when it formats a disk, holds information about the location of each file as it is stored. See also New Technology file system (NTFS), high-performance file system (HPFS).

file buffer

A reserved portion of memory used to temporarily store data, pending an instruction to complete its transfer to or from a file.

file exception

An exception (abnormal condition or error) that occurs during the course of opening, reading from,

or writing to a file. See also C++ exception handling.

file handle

A unique identifier that Windows assigns to a file when the file is opened or created. A file handle is valid until the file is closed.

file input/output (I/O)

The mechanism for making data persistent between program work sessions by creating files, reading from files, and writing to files.

file pointer

A pointer that specifies the next byte to be read or the location to receive the next byte written in a file.

file scope

Or global scope. The degree of visibility of an identifier (C/C++ name) when it is declared outside all blocks or classes. The identifier is accessible anywhere in the translation unit after its declaration. See also class scope, function scope, function-prototype scope, local scope, external linkage.

file status

Information on whether the file exists, its creation and modification dates and times, its logical size in bytes, its attributes, and its path. See also file time.

file time

A 64-bit value that gives the file creation time, last access time, or last write time, as represented by the number of 100-nanosecond intervals that have elapsed since January 1, 1601. Windows records each file time in coordinated universal time (UTC) format. See also file status, system time.

File Transfer Protocol

A method of retrieving files to your home directory or directly to your computer using TCP/IP. Many Internet sites have established publicly accessible repositories of materials that can be obtained using FTP with the account name anonymous. Thus, these sites are called anonymous ftp servers.

filename extension

In the MS-DOS 8.3 filename convention, an optional period (.) followed by up to three characters that can be appended to the eight-character base filename. See also base name.

final class

In Java, a class that can have no subclasses.

final method

In Java, a method that cannot be overridden.

final variable

In Java, a variable whose value cannot be changed. Corresponds to the C++ constant.

firewall

Or proxy server. A system or combination of systems that enforces a one-way barrier between two or more networks, usually used for security purposes. Firewalls accomplish all communication between the network and outside.

fixup

1 Or relocation information. Information generated by the linker for addresses that cannot be determined at link time. For example, the linker creates fixups for addresses within the executable that are relative to the executable's base address and for addresses of dynamically loaded DLLs. The Windows loader uses this information to resolve these addresses when the program or DLL is loaded.

2 A record (FIXUPP) generated by the assembler for each address it cannot determine (for example, addresses of external symbols). The fixup contains the information the linker will need to determine the address.

flag

Broadly, a marker of some type used by a computer in processing or interpreting information. Such a signal indicates the existence or status of a particular condition. Depending on its use, a flag can be code, embedded in data, that identifies some condition, such as the beginning or end of a word or a message, or it can be one or more bits set internally by hardware or software to indicate an event of some type, such as an error or the result of comparing two values.

floating toolbar

A toolbar that can appear anywhere on the user's display and is always on top of all other windows. Its size or position can be modified when floating. See also dockable toolbar, docked toolbar.

floating type

A general category of arithmetic data types that are capable of storing a floating-point value (a number that may have a fractional part). The data types **float**, **double**, and **long double** are floating types. See also integral type.

floating-point function

A function that returns a value defined as a floating-point type—that is, a **float**, **double**, or **long double**. See also floating-point number.

floating-point number

Or real number. A value that may contain an integer component, a fractional component, or both. In C/C++, the data types **float**, **double**, and **long double** can store floating-point numbers.

flow control

In data communications, a mechanism to prevent the sender from transmitting data faster than the receiver can handle the incoming data. Flow control usually depends on a set of protocols established at the beginning of the transmission session that define how and when a sender may transmit.

focus

A temporary property of a user-interface object, such as a window, view, dialog box, or button, that permits the object to receive keyboard input from the user. The focus is usually conveyed through highlighting. See also top-level window.

font mapper

In Windows, an operating system component used to find the physical font that most closely matches a specified logical font. The font mapper uses an internal algorithm that compares the attributes of the requested logical font against the attributes of available physical fonts. This mapping occurs when the font is actually used for the first time. See also logical font, physical font.

font resource

A group of individual fonts representing characters in a given character set that have various combinations of heights, widths, and pitches. The Windows operating system maintains a font table containing all the fonts that applications can use. You can load font resources and add the fonts in each resource to this table by using the **AddFontResource** Windows function.

foreground color

The color that is currently selected for drawing or displaying text on screen. In monochrome displays, the foreground color is the color of a bitmap or other graphic. See also background color.

foreground window

Or active window. The window with which the user is currently working. The foreground window is identified by color changes to the title bar and border. See also topmost window, top-level window, focus.

form view

A program window whose client area contains dialog-box controls to permit entering, viewing, or altering data, generally in a form-based data-access application.

form-based application

An application whose user interface is based on a form containing controls in which a user examines, enters, or edits data.

formal argument

Or formal parameter. An argument that is declared in the function header and used in the body of a function. Calling functions pass values to called functions in actual arguments. The called function accesses the values using its corresponding formal arguments.

formal parameter list

The parameters specified in a particular method or function definition. Contrasts with the actual parameter list, which appears in the declaration.

format

1 As a noun, the structure or appearance of a unit of data, such as a file, fields in a database record, a cell in a spreadsheet, or the text in a word-processing document.

2 As a verb, to change the appearance or organization of the selected material. To format a disk is to prepare a disk for use by organizing its storage space into a collection of data compartments, each

capable of being addressed by the operating system.

format string

A string that can contain specifications for various kinds of type formats as well as literal characters. For example, in the C statement

```
printf( Total Expenses:  $%.2f \n,  Sum );
```

the format string (enclosed in double quotation marks) contains the literal string `Total Expenses: $`, the formatting characters `%.2f` to print the value of `Sum` as a decimal number with two digits to the right of the decimal point, and the `\n` character to begin a new line.

formatting rectangle

In Windows, a construct for formatting the text displayed in the window rectangle. An application can make the formatting rectangle larger than the window rectangle (limiting the visibility of the edit control's text) or smaller than the window rectangle (thereby creating extra white space around the text).

forward reference

In C/C++, a reference to a class, variable, or function that has been declared but not yet defined.

frame window

In MFC, the window that coordinates the interactions of the application with a document and its view. The frame window provides a visible frame around a view, with an optional status bar and standard window controls such as a control menu, buttons to minimize and maximize the window, and controls for resizing the window. The frame window is responsible for managing the layout of its child windows and other client-area elements such as control bars and views. The frame window also forwards commands to its views and can respond to notification messages from control windows. In OLE, a frame window is the outermost main window where the container application's main menu resides. See also main frame window.

framework

See application framework.

Frequently Asked Questions

Documents that list and answer the most common questions on a particular subject.

friend

A keyword used within a class declaration to specify that a function or another class has access to the private and protected members of the first class. The function or class specified with the **friend** keyword is considered a friend of the first class.

FTP

A method of retrieving files to your home directory or directly to your computer using TCP/IP. Many Internet sites have established publicly accessible repositories of materials that can be obtained using FTP with the account name anonymous. Thus, these sites are called anonymous ftp servers.

full link

A non-incremental build of program files in which an incremental status (.ILK) file is generated. The

incremental status file has the same base name as the executable (.EXE) file or dynamic-link library (.DLL) file targeted by the link, and a .ILK filename extension. During subsequent incremental builds, the linker uses and updates the incremental status file. See also incremental link.

full-server application

In OLE, an application that can be run either as a stand-alone application or launched by a container application. A full-server application can store documents as files on disk and supports both embedding and linking. See also mini-server application, server application.

fully qualified path

Or absolute path. The location of a file or directory on a volume, including a drive letter and any intervening directory names. See also relative path.

function

- 1 A block of code, consisting of a return type, function name, optional parameters, and statements, that performs one or more specific tasks within the source program and returns a value to the caller.
- 2 The purpose of or the action carried out by a program, routine, or other object.

function body

The portion of a function definition that contains the declarations of its local variables and executable statements. See also function declaration.

function call

A postfix expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the actual arguments to the function.

function counting

A run-time analysis of a program in which the profiler records how many times each function was called, which is its hit count. See also function profiling, function timing, function coverage, line counting.

function coverage

A run-time analysis of a program in which the profiler reports whether a function was called. This analysis shows which sections of code (functions) are not being executed. See also function counting, function profiling, function timing, line coverage.

function declaration

A statement consisting of a return type, followed by the function name, followed by a list of the names and types of formal parameters enclosed in parentheses. In C++, a function must be declared before it can be called.

function definition

Specifies the name of the function, the types and number of parameters it expects to receive, and its return type. A function definition also includes a function body with declarations of the function's local variables and the statements that determine what the function does. Function definitions differ from function declarations in that they supply function bodies—the code that makes up the function.

function overloading

In C++, specifying more than one function of the same name but with different parameters, in the same scope. These functions, called overloaded functions, enable programmers to supply different semantics for a function, depending on the types and number of arguments. For example an overloaded function could be called `print`, regardless of whether it printed a single string, a list of integers, or all the data members of a class. The arguments supplied in the function call determine which `print` function is called. See also function overriding.

function overriding

In C++, redefining a member function of a base class from within a derived class. An overriding function has exactly the same name, parameters, and return type as the overridden function. See also function overloading.

function profiling

A run-time analysis of code execution by function, in which the profiler detects inefficiencies by counting and timing functions. See also function counting, function coverage, function timing, line profiling.

function prototype

Names the function and its parameters (if any), and provides type information for the return value and parameters (if any). See also formal parameter list.

function scope

In C/C++, the degree of visibility of a label. Labels are accessible only within the function where they are declared. See also class scope, file scope, function-prototype scope, local scope.

function set

One or more functions or data objects that can be exported by a shared library so that they are available to other programs.

function template

- 1 A mechanism for specifying a set of functions that are based on the same code but act on different types or classes. When a templated function is first called for each type, the compiler creates an instantiation, a specialized version of the templated function for the type.
- 2 A complete member function definition with an empty function body that ClassWizard writes in the source files that contain the class of which the member is a function.

function timing

A run-time analysis of a program in which the profiler records how many times each function was called (the hit count) as well as how much time was spent in each function and any called functions. See also function counting, function coverage, function profiling.

function-prototype scope

Or prototype scope. In C++, the degree of visibility of an identifier when it is declared within a function prototype. The identifier is accessible only to the function declarator (delimited by parentheses). Such identifiers are effectively comments, to make the parameters of the prototyped function readily apparent to a reader. See also class scope, file scope, function scope, local scope.

functionality

The features, operations, functions, or capability supported by a program or program component. Some examples of common functionality are serialization, in-place activation, OLE, and ODBC.

fundamental type

A data type that is built into the language. In C/C++, fundamental types can be divided into three categories: integral, floating, and void. Integral types are capable of handling whole numbers. Floating types are capable of specifying values that have fractional parts. The **void** type describes an empty set of values.

Glossary G

garbage collector

A process that periodically frees the memory used by objects that are no longer needed.

gateway

A host computer that connects networks that communicate using different protocols. For example, a gateway connects a company's local area network to the Internet.

GDI

An executable program that processes graphical function calls from a Windows-based application and passes those calls to the appropriate device driver, which performs the hardware-specific functions that generate output. By acting as a buffer between applications and output devices, GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device.

general protection fault

An exception that is raised when an application attempts to read from or write to an area of memory that is not owned by the application. Usually, an application must be terminated after a general protection fault.

GIF

A form of graphics compression. See also Joint Photographic Experts Group (JPEG).

global

Universal, in the sense of being related to an entire file, document, program, or other entity. For example, a global variable is one that is accessible from anywhere in the program.

global variable

A variable that is accessible from anywhere in a program. A global variable has storage and maintains a value throughout the program's execution. See also file scope.

globally unique identifier

See universally unique identifier.

glyph

The bitmap, collection of points, or collection of graphic commands that define a single character or symbol in a font. See also bitmap.

gopher

A client/server application that allows the user to browse large amounts of information. It presents the information to the user in a menu format. When capitalized, it refers to the original Gopher server developed at the University of Minnesota.

GPF

An exception that is raised when an application attempts to read from or write to an area of memory

that is not owned by the application. Usually, an application must be terminated after a general protection fault.

graphic object

One of the drawing tools, such as a pen, brush, bitmap, palette, and so on, that Windows provides for use in device contexts.

graphics device interface

An executable program that processes graphical function calls from a Windows-based application and passes those calls to the appropriate device driver, which performs the hardware-specific functions that generate output. By acting as a buffer between applications and output devices, GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device.

Graphics Interchange Format

A form of graphics compression. See also Joint Photographic Experts Group (JPEG).

group-box control

A labeled rectangle used to define a group of related controls (usually check boxes or radio buttons) in a dialog box. The group box itself does not set any options; it is simply a graphical device used to improve the usability of complex dialog boxes. See also radio group.

guarded body of code

A set of one or more statements for which an exception or termination handler provides protection.

GUID

See UUID.

Glossary H

handler

- 1 In OLE, a DLL that resides in a client process and performs some tasks on behalf of the server, while delegating other tasks back to the server itself.
- 2 In general, a routine that manages a common and relatively simple condition or operation, such as error recovery or data movement. See also message handler.

hash value

The value of a key that has been numerically manipulated to directly calculate either the location of its associated record in a table or the starting point for a search for the associated record. If the key value is a character string, each possible character is assigned a numeric code to permit the numerical manipulation. The manipulation performed on the key value is known as the hashing function.

head

- 1 The beginning of an item, such as a list, nonscrolling region, string, or transmission queue.
- 2 In hardware, the read/write mechanism in a disk drive or magnetic tape drive, or the printing mechanism in a printer.

header control

In MFC, a window usually positioned above columns of text or numbers. The header control contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column.

header file

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

heap

- 1 Or free store. A portion of memory reserved for a program to use for the temporary storage of data structures whose existence or size cannot be determined until the program is running. The program can request free memory from the heap to hold such elements, use it as necessary, and later free the memory.
- 2 In sorting, a partially ordered complete binary tree.

heap allocation

Reserving memory for the needs of the program. See also heap, stack.

Help context

A string and a number (Help context ID) that an application passes during a call to Windows Help in order to locate and display a Help topic. See also Help map file, Help project file.

Help file

A file that contains text and graphics needed to communicate online information about an application. Each help file contains one or more topics a user can select by clicking hot spots, using the

keyword search, or browsing through topics. Help files have a .HLP filename extension. See also Help topic.

Help map file

A file that defines Help context IDs corresponding to the IDs of dialog boxes, menu commands, and other resources in an application. The AppWizard file MAKEHELP.BAT calls the MAKEHM tool to generate this file from the contents of a RESOURCE.H file. The Help map file has a .HM filename extension. See also Help project file.

Help project file

A project file that controls how the Windows Help Compiler creates a Help (.HLP) file from topic files. The Microsoft Help Workshop is used to create a Help project file. The filename extension of a Help project file is .HPJ.

Help topic

The primary unit of information in a Help (.HLP) file. A topic is a self-contained body of text and graphics, similar to a page in a book. Unlike a page, however, a topic can hold as much information as you require. If there is more information in a topic than the Help window can display, scroll bars appear to let the user scroll through the information.

hexadecimal

Or hex. The base-16 counting system, whose digits are 0 through F. The letters A through F represent the decimal numbers 10 through 15. Two hex digits are needed to represent 1 byte. See also binary.

hidden text

In a document file, special characters that are not normally printed or displayed on the screen. For example, in a rich-text format (.RTF) file for a Help topic, the following hidden text string generates a jump to another Help topic:

```
Jump textcontextstring@d:\path\file.hlp
```

high order

In a group of bits or bytes, the one that carries the most weight or significance. Within a byte, the leftmost bit is the high-order bit. Within a group of bytes, the byte-ordering convention (big-endian or little-endian) determines the position of the high-order byte.

high-performance file system

A fixed-disk file system that organizes the disk into volumes, rather than partitions and logical drives. HPFS supports long, mixed-case filenames and extended attributes, and implements several levels of caching, for improved operating-system performance. The IBM OS/2 operating system supports HPFS. See also file allocation table file system, New Technology file system (NTFS).

highlighting

Altering the appearance of displayed characters as a means of calling attention to them—for example, by displaying them with higher intensity or by using reverse video (dark on light instead of light on dark, or vice versa). Highlighting is often used to denote characters to be deleted, copied, or otherwise acted upon.

hit count

The number of times a function or line of source code is called (or hit), as reported by the profiler when analyzing a program at run time. See also function profiling, line counting, line profiling.

hit-test code

A code that determines the location of the cursor. For example, HTLEFT means the cursor is in the left border of the window; HTMAXBUTTON means the cursor is in a Maximize button.

home page

The main page of a Web site as generated by the developer of the site. Although the home page is often the first page a visitor to the site sees, it is not the same as start page.

hook

- 1 A point in the Windows message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of messages before they reach the target window procedure.
- 2 More generally, a location in a routine or program at which the programmer can connect or insert other routines for the purpose of debugging or enhancing functionality.

hook procedure

An application-installed procedure that monitors the system for events associated with either a specific thread or all threads in the system. For example, the hook code WH_GETMESSAGE installs a hook procedure that monitors messages posted to a message queue. See also callback function.

hot key

- 1 In Windows, an application-defined key combination used to obtain high-priority keyboard input from the user. For example, an application can have a hot key that allows the user to cancel a lengthy operation.
- 2 A key combination that the user can create to perform an action quickly. See also accelerator key.

hot spot

The pixel in a cursor that marks the exact screen location affected by a mouse action, such as a button click. Mouse messages include the coordinates of the hot spot.

hourglass cursor

A cursor displayed in the form of an hourglass that indicates that the program is performing a lengthy task. The cursor can be displayed as a large hourglass, or as a smaller hourglass with an arrow.

HPFS

A fixed-disk file system that organizes the disk into volumes, rather than partitions and logical drives. HPFS supports long, mixed-case filenames and extended attributes, and implements several levels of caching, for improved operating-system performance. The IBM OS/2 operating system supports HPFS. See also file allocation table file system, New Technology file system (NTFS).

HTML

A markup language derived from SGML. Used to create a text document with formatting specifications that tells a software browser how to display the page or pages included in the document.

HTTP

The Internet protocol used by World Wide Web browsers and servers to exchange information. The protocol makes it possible for a user to use a client program to enter a URL (or click a hyperlink) and retrieve text, graphics, sound, and other digital information from a Web server. HTTP defines a set of commands and uses ASCII text strings for a command language. An HTTP transaction consists of a connection, a request, a response, and a close.

HTTP server

Or Web server. A server that runs a Hypertext Transfer Protocol service or application. This protocol defines the procedures used when connecting a Web browser to a Web server.

hyperlink

The means used to jump to another Web page. It consists of both the display text the user sees and the URL of the reference. See also hypertext link.

hyperlink address

The path to an object, document, or page. A hyperlink address can be a URL or a UNC network path. Can also contain display text or sublocation information (for example, a database object, Word bookmark, or Microsoft Excel cell range to which the address points).

hyperlink base

A partial path based on a user-defined path. Use a hyperlink base if you want to move either the file that contains the hyperlink or the destination file separately and still maintain the hyperlink.

hypertext link

In a Web document, this is a reference to another Web document. The user does not actually see a URL in the document; instead, a highlighted reference contains a pointer to a link. The user clicks on the highlighted reference and the desired Web page is automatically retrieved. See also hyperlink.

Hypertext Markup Language

A markup language derived from SGML. Used to create a text document with formatting specifications that tells a software browser how to display the page or pages included in the document.

hypertext reference

Attribute of the HTML anchor element that identifies the anchor as a hyperlink. Its value determines the destination of the hyperlink.

Hypertext Transfer Protocol

The Internet protocol used by World Wide Web browsers and servers to exchange information. The protocol makes it possible for a user to use a client program to enter a URL (or click a hyperlink) and retrieve text, graphics, sound, and other digital information from a Web server. HTTP defines a set of commands and uses ASCII text strings for a command language. An HTTP transaction consists of a connection, a request, a response, and a close.

Glossary I

ICMP

An extension to the Internet Protocol that allows for the generation of error messages, test packets and informational messages related to IP.

icon file

In Windows, a file that contains a bitmap of an icon. Icon files usually have a .ICO filename extension.

icon resource

An icon that is stored in an application's resource-definition file. At run time, an application can call the **LoadIcon** Windows function to retrieve the handle of the icon. Icon resources can be used to avoid device dependence, simplify localization, and enable applications to share icon shapes.

idle state

The state of a modal dialog box or menu when it has finished processing a message and it has no more messages waiting in its active message queue.

idle time

The period during which the application has an empty message queue. Idle time permits the processing of background tasks.

IEEE

A professional organization of computer hardware and software engineers. The institute has developed standards for many aspects of computer technology, such as network connectivity, and formats for representing floating-point numbers.

IETF

The primary working body developing standards for the Internet.

image list

In MFC, a collection of same-sized images contained in a single, wide bitmap. Image lists are used to efficiently manage large sets of icons or bitmaps.

immediate rendering

In a data-transfer operation, making data immediately available to an application, versus making it available when requested (delayed rendering).

impersonation token

In Windows NT, an access token that has been created to capture the security information of a client process, allowing a server to impersonate the client process in security operations. See also primary token, privilege, security identifier (SID).

implementation

A description of the data structure used to represent an object's core state, definitions of the methods that can access the data structure, and information about the intended type of the object. Can also refer to a single function, where the function definition provides the implementation for that function.

implementation file

In MFC, the source code file (usually a C++ source file with a .CPP or .CXX filename extension) that contains a single class definition along with the code that implements that class's member functions. See also interface file.

implementation-defined

Behavior that depends on the implementation, with the range of possible behaviors often delineated by a standard.

import file

A file that describes the functions and data to be exported when an export (.EXP) file is referenced by a linked program. The Incremental Linker (LINK.EXE) uses the export file to build a program that contains exports (usually a dynamic-link library), and it uses the import library to resolve references to those exports in other programs. Import library files usually have a .LIB filename extension. See also library file.

imported function

One called inside another executing entity, such as a DLL or executable file. Functions can be imported either through use of a .LIB import library, or by using the __declspec (dllimport) storage-class modifier, a Microsoft-specific extension to the C language. This modifier explicitly defines the client's interface to other services. See also exported function.

in-memory file

A file that behaves like a disk file except that its bytes are stored in RAM. An in-memory file is a useful means of transferring raw bytes or serialized objects between independent processes.

in-place activation

Or in-place editing, visual editing. The ability to activate an object within the context of its container document, as opposed to opening it in a separate window.

increment (++) operator

An operator that specifies that its integral or floating-point operand be increased by the integer value 1. An incremented pointer points to the next object. In the prefix form (++i), the increment takes place before the value is used in expression evaluation; in the postfix form (i++), the increment takes place after the value is used in expression evaluation. See also decrement (--) operator.

incremental link

The process of rebuilding only those program files that have a timestamp that is more recent than the last link, or updating only those parts of the program that have changed. An incrementally linked program is functionally equivalent to a program that is linked nonincrementally, but it differs from a nonincremental program in that it is prepared for subsequent incremental links and is larger as a result. See also incremental status file, full link.

incremental status file

A state file generated to hold status information for later incremental links of the program. The file has the same base name as the executable file or dynamic-link library and the filename extension .ILK. The incremental status file is created the first time the Incremental Linker (LINK.EXE)

runs in incremental mode. LINK updates the file during subsequent incremental builds. LINK is the only tool that uses the .ILK file. See also incremental link.

Indexed Sequential Access Method

Pronounced EYE-sam. A scheme for decreasing the time necessary to locate a data record within a large database, given a unique key for the record. The key is the field in the record used to reference the record.

indirect memory operand

In an assembly-language instruction, a memory operand whose value is treated as an address that points to the location of the desired data.

indirection (*) operator

Or dereferencing operator. In C/C++, a unary operator (*) that accesses a value indirectly, through a pointer. The operand must be a pointer value. The result of the operation is the value stored at the address to which the operand points. If the operand points to a function, the result is a function designator. If it points to a storage location, the result is an l-value designating the storage location. See also address-of (&) operator.

infix notation

Placing of operators between operands, as in $(x+y)*z$. C++ and Java binary operators use infix notation exclusively. Compare prefix notation, postfix notation.

inheritance

In object-oriented programming, a method for deriving new classes from existing classes. The derived class inherits the description of its base class(es), but can be extended by adding new member variables and functions and by using virtual functions. A class can inherit from a single base class (single inheritance) or from any number of direct base classes (multiple inheritance). A class derived using multiple inheritance has the attributes of all of its base classes.

inheritance hierarchy

A classification of items in which each item except the top one (known as the root) is a specialized form of the item above it. Each item can have one or more items below it in the hierarchy. In the Java class hierarchy, the root is the **java.lang.Object** class. In MFC, the root is the **CObject** class. See also root.

initialization file

In Windows, a file that an application uses to store information that otherwise would be lost when the application closes. Initialization files typically contain information such as user preferences for the configuration of the application. Initialization files usually have a .INI filename extension.

initializer

The portion of a declarator that specifies an initial value for an object or a variable.

inline assembler

1 A feature of the Visual C++ compiler that allows the use of assembly-language instructions in C/C++ source programs without extra assembly and link steps. The inline assembler is built into the compiler; therefore, a separate assembler is not needed. Inline assembly code can use any C/C++

variable or function name that is in scope.

2 Assembly-language code that is inserted into C/C++ source code. The `_asm` keyword preceding an assembly-language statement or block of statements invokes the inline assembler at compile time.

inline file

A file that contains text specified in the makefile. The name of file can be used in commands as input (for example, a LINK command file), or it can pass commands to the operating system. The file is created on disk when a command that creates the file is run.

inline function

1 In C++, a function defined in the body of a class declaration. Inline functions are typically one- or two-line functions used to return information about an object's state.

2 A function whose declaration is preceded by the keyword **inline**, instructing the compiler to replace calls to that function with the code of the function body. This substitution occurs only at the compiler's discretion. For example, the compiler does not expand a function inline if its address is taken or if it is too large to expand inline.

insertion (<<) operator

Or output operator, put-to operator. In C++, the left-shift operator, overloaded (in the iostream library) to provide formatted output. The left operand must be one of the predefined output streams (**cout** or **cerr**) and the right operand can be any valid C++ expression. See also extraction (>>) operator, overloaded operator.

inside-out

In OLE, a model for in-place objects (for example, windows) that the user can activate with a single click. An inside-out object remains visible to the user when its user interface is deactivated. See also outside-in.

instance

1 In general, an object of a particular class.

2 In Java, an instance of a class is created using the **new** operator followed by the class name.

3 In C++, an instance can be created by defining it on the stack. In this case, the **new** keyword is not used. You can also instantiate an OLE object by calling the OLE API **CoCreateInstance**.

instance handle

A handle that Windows assigns to each copy of a loaded application or dynamic-link library (DLL) in a multitasking system. Every window class requires an instance handle to identify the application or DLL that registered the class. See also task handle.

instance method

Any method that can be invoked using an instance of a class, but not using the class name. Instance methods are defined in class definitions. Called nonstatic member functions in C++. See also class method.

instance variable

Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Called member variables in C++. See also class variable.

Institute of Electrical and Electronics Engineers

A professional organization of computer hardware and software engineers. The institute has developed standards for many aspects of computer technology, such as network connectivity, and formats for representing floating-point numbers.

integral type

A general category of arithmetic data types that are capable of storing an integer (whole number). The data types **char**, **short**, **int**, **long**, and **enum** are integral types. The optional keywords **signed** and **unsigned** can precede or follow any of the integral types except **enum**, or these keywords can also be used alone as type specifiers, in which case they are understood as **signed int** and **unsigned int**, respectively. See also floating type.

Integrated Services Digital Network

A type of digital communications service offered by telephone companies. It can carry data, voice, and video over specially conditioned high-speed telephone lines delivering two 64,000 bps bearer channels and one 16,000 bps data signaling channel.

interface

- 1 In the Component Object Model, a set of related functions; a description of an abstract type.
- 2 In Java, a group of methods that can be implemented by several classes, regardless of where the classes are in the class hierarchy.
- 3 An IDL keyword used by the MIDL compiler for generating interface declarations.

interface file

In MFC, the source header (.H) file that contains a single class declaration and any other information needed to use the class. See also implementation file.

internal linkage

Specifies that the names of objects and functions should refer only to program elements inside their own translation units. The names are not shared with other translation units. The keyword **static** before a name declaration ensures internal linkage. See also external linkage.

Internet

A collection of computer networks that connects millions of computers around the world.

Internet Control Message Protocol

An extension to the Internet Protocol that allows for the generation of error messages, test packets and informational messages related to IP.

Internet Engineering Task Force

The primary working body developing standards for the Internet.

Internet Protocol

The basic protocol of the Internet. It enables the delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability.

Internet Relay Chat

A multi-user system where people convene on channels (a virtual place, usually with a topic of conversation) to talk in groups or privately on the Internet.

Internet server application

Internet server extension DLL.

Internet Server Application Programming Interface

A set of functions for Internet servers, such as a Windows NT Server running Microsoft Internet Information Server (IIS).

Internet server extension DLL

Or Internet server application (ISA). A DLL that can be loaded and called by some HTTP servers. Used to enhance the capabilities of applications that extend a Web server.

Internet server extensions

An application that processes server requests, including Common Gateway Interface (CGI) and ISAPI applications.

Internet server filter

A routine that receives notification of server events such as URL mapping and logon requests, and filters, examines, or changes data to and from the browser or Web server.

Internet Service Provider

A company that provides access to end users of the Internet, as opposed to Network Service Providers (NSP's).

interpreter

A program that can execute code that is not native to the machine it is being run on. Java programs are often run by an interpreter that decodes the bytecodes that make up the program. See also JIT compiler.

intranet

A network within an organization, usually connected to the Internet via a firewall, that uses protocols such as HTTP or FTP to enhance productivity and share information.

intrinsic function

1 In certain high-level programming languages, such as FORTRAN, a function that is part of the language. The compiler automatically links intrinsic functions to the program without any additional effort on the programmer's part. Programs that use intrinsic functions are faster because they do not have the overhead of function calls, but they may be larger due to the additional code generated.

2 In Microsoft C++, a library function, such as **strcmp** or **strcpy**, that has an intrinsic form. Use of the `#pragma intrinsic` compiler directive generates these functions as inline code rather than as function calls.

IP

The basic protocol of the Internet. It enables the delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability.

IP address

The Internet protocol address which is a 32-bit address assigned to a host. The IP address has a host component and a network component.

IP number

An Internet address that is a unique number consisting of 4 parts separated by dots, sometimes called a dotted quad (for example, 198.204.112.1). Every Internet computer has an IP number and most computers also have one or more domain names that are plain-language substitutes for the dotted quad.

IRC

A multi-user system where people convene on channels (a virtual place, usually with a topic of conversation) to talk in groups or privately on the Internet.

ISA

Internet server extension DLL.

ISAM

Pronounced EYE-sam. A scheme for decreasing the time necessary to locate a data record within a large database, given a unique key for the record. The key is the field in the record used to reference the record.

ISAPI

A set of functions for Internet servers, such as a Windows NT Server running Microsoft Internet Information Server (IIS).

ISAPI filter

An Internet server filter packaged as a dynamic-link library that runs on ISAPI-enabled servers.

ISDN

A type of digital communications service offered by telephone companies. It can carry data, voice, and video over specially conditioned high-speed telephone lines delivering two 64,000 bps bearer channels and one 16,000 bps data signaling channel.

ISO/OSI

International Organization for Standardization/Open Systems Interconnection

ISP

A company that provides access to end users of the Internet, as opposed to Network Service Providers (NSPs).

Glossary J

Java

A programming language similar to C++. Java is currently popular because it is compiled to machine-independent bytecode. This allows programmers to write one kind of program code for all platforms (Macintosh, Windows 3.x, Windows NT, Windows 95, UNIX). Software components written in Java for the Web are called applets. Java applets access code libraries on local clients and can download additional class files from the server.

JIT compiler

Just-In-Time compiler for Java. The JIT compiler takes the Java bytecode (which is machine-independent) and compiles it on demand into native code for the target machine, giving faster execution. Since JIT compilers operate on the client machine, they preserve the platform-independence of the compiled Java program.

join

A database operation that combines records from two or more tables, based on an exact match of key values in these tables. Once a recordset has been created based on a join, it functions as if the records were all in one table.

Joint Photographic Experts Group

An algorithm for graphics file storage, supported by many Web browsers. JPEG was developed for compressing and storing photographic images and is best used for graphics containing many colors, such as scanned photos. Most files that are compressed using JPEG are stored in a file format called JFIF (JPEG File Interchange Format). See also Graphics Interchange Format (.GIF).

JPEG

An algorithm for graphics file storage, supported by many Web browsers. JPEG was developed for compressing and storing photographic images and is best used for graphics containing many colors, such as scanned photos. Most files that are compressed using JPEG are stored in a file format called JFIF (JPEG File Interchange Format). See also Graphics Interchange Format (.GIF).

jump

A word, phrase, or button in a Help topic that, when clicked, takes the user to another topic screen in the Help system.

jump statement

A statement that either transfers control immediately to another location in the function or returns control from the function. In C++, the jump statements are **break**, **continue**, **return**, and **goto**.

Glossary K

kernel

The core component of the operating system. The kernel schedules activities (threads) for the computer processor to perform, and it handles interrupts and exceptions. If the computer has multiple processors, the kernel synchronizes activity among the processors to optimize performance.

key value

A field or expression used to identify a record; often used as the index field for a database table.

keyboard accelerator

See accelerator key.

keyword

Or reserved word. A word that has special meaning to a program or in a programming language. Reserved words usually include those used for control statements, data declarations, and the like. A reserved word can be used only in certain predefined circumstances; it cannot be used in naming functions, variables, structures, labels, classes, or user-generated tools such as macros.

Glossary L

l-value

The value on the left side of an assignment statement that represents a storage region's memory location. The l-value is an expression that evaluates to a type other than **void** and designates a location in memory (a variable, array, or structure element, for example). See also r-value.

label

- 1 In a C or C++ function body, a unique name followed by a colon (:). Labels can denote statements to which a **goto** statement can branch. In a **switch** statement, the labels preceded by the keyword **case** list values to be compared to the expression at the top of the **switch** statement. See also labeled statement.
- 2 In user-interface design, application-defined text associated with a graphical element such as a button or check box.
- 3 In OLE, text describing an object that is linked or embedded as an icon.

labeled statement

In C/C++, a statement preceded by a label. A labeled statement allows program control to be transferred to it from a **goto** statement or back to the top of a **switch** statement.

language identifier (ID)

A 16-bit value that identifies a language and, where appropriate, the variant of the language being used. A language identifier is a combination of a sublanguage identifier and a primary language identifier. For example, if the primary language identifier specifies English, the sublanguage identifier might specify Australian English. See also locale identifier (LCID).

LCID

A 32-bit value identifying the language and sublanguage for a locale. The locale identifier is used to customize string handling. See also code page, language identifier (ID).

lead byte

In double-byte character sets and multibyte character sets, the first byte of a two-byte character. The lead byte signals that both it and the following byte are to be interpreted as a single character. See also trail byte.

left outer join

A database term that describes the relationship between two tables on which a query is run. If two tables are connected with a left outer join, the resulting recordset contains all records from the table on the left but only those records from the table on the right where the joined fields are equal. See also right outer join.

lexical analyzer

Or lexer. The part of a compiler, interpreter, or other translator that parses character sequences (separated by white space) in the source code into individual tokens. See also parser.

library assert

A macro that identifies program errors during development. The argument given to ASSERT should be chosen so that it holds true only if the program is operating as intended.

library file

A Common Object File Format (COFF) file generated by the Microsoft 32-bit library manager tool, LIB, for standard and import libraries. The default filename extension for these files is .LIB. See also dynamic-link library (.DLL), static-link library.

library version

1 One of the available configurations of a library. For example, a library might have a static version and a dynamic-link version, both of which are available as debug and release versions. The library version used during linking is usually determined by the project type and settings.

2 One of the publicly released versions of a library—for example, Microsoft Foundation Class Library version 4.0.

lifetime

The time during program execution that a variable, function, object, or session exists and is available for use, or the duration of the program itself.

Lightweight Remote Procedure Call

In OLE, a protocol for interprocess communication on a single machine. See also Remote Procedure Call (RPC).

line counting

A run-time analysis of a program in which the profiler reports how many times each line of source code was executed. See also function counting, line coverage, line profiling.

line coverage

A run-time analysis of a program in which the profiler reports which lines of source code were executed at least once. See also function coverage, line counting, line profiling.

line profiling

A run-time analysis of a program in which the profiler reports information about the execution of the lines of source code, including line counting and line coverage. Line profiling is useful for checking the validity of an algorithm. See also function profiling.

line-continuation character

The backslash character (\) when it is placed at the end of a line. The line-continuation character causes the compiler to ignore the following newline character and to treat the next line as a continuation of the current line.

linefeed character

A control character that advances a printer or insertion point to a new line. This character can have a different textual representation on different platforms, but it always has the ASCII value of 10. See also carriage return–linefeed (CR-LF) pair.

link

- 1 The Microsoft 32-Bit Incremental Linker (LINK.EXE). See also linker.
- 2 To combine object files and libraries to form an executable file or dynamic-link library.
- 3 In OLE, a connection between two documents. A link has three properties: the name of its source data, its type (or class, as it is known internally), and its updating basis (either automatic or manual). Also, as a verb, it means to connect two documents with a link.

link time

The period of time required by the linker to combine (link) object files and libraries during compilation of a program, or the point in time when these files are combined. See also linker.

linkage specification

The protocol for linking functions (or procedures) written in different languages. Function calling conventions are affected by the linkage specification selected. An example of a linkage specification is **extern C**. See also external linkage, internal linkage.

linked item

Or linked object. In OLE, an item in a compound document whose data is stored in a separate file rather than in the document's file. A linked item must be edited in a separate window. See also embedded item.

linked list

In programming, a data structure consisting of nodes or elements connected by pointers. A singly linked list has one pointer in each node, pointing to the next node in the list; a doubly linked list has two pointers in each node, pointing to the next and previous nodes. In a circular list, the first and last nodes of the list are linked together.

linker

A utility that combines object files and libraries to create an executable file or dynamic-link library. During the linking process, the linker resolves external references such as a call to a library routine by the program.

linking and embedding

Two methods available in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file. See also Object Linking and Embedding (OLE).

list view control

A Windows Common Control that displays a collection of items each consisting of an icon and a label.

list-box control

In Windows, a child window that contains a list of items that can be selected by the user. List boxes can permit the selection of one item or multiple items. See also combo-box control.

listserv

An Internet application that automatically serves mailing lists by sending electronic newsletters to a stored database of Internet user addresses. Although some listservs are moderated, users can often

handle their own subscribe/unsubscribe actions without requiring anyone at the server location to personally handle the transaction.

literal

A value, used in a program statement, that is expressed directly rather than as a named constant or the contents of a variable. For example, in the statements

```
i = 25;  
c = 'a';  
cout << Hello;
```

the values 25, 'a', and Hello are literals. See also manifest constant, string literal.

little-endian

One of two byte-ordering conventions used on different machines. In little-endian addressing, the address points to the least significant byte of the word. Intel 80x86 and DEC RISC computers are little-endian machines. See also big-endian.

load time

The period of time required to place a program's executable files into memory prior to execution, or the point in time when the files are loaded.

loaded state

In OLE, the status of a compound-document object (either a linked or embedded item) whose handler is loaded in memory but whose server is not running. See also running state.

local

- 1 In programming, describes an object, memory location, or variable whose scope or lifetime is limited.
- 2 A Microsoft Windows NT keyword that identifies the local attribute, which can be applied to individual functions or to the interface as a whole.
- 3 In distributed systems, describes an operation that is performed by the current application rather than by another application on the same computer or on a remote computer.
- 4 In networking, describes a device that can be accessed directly rather than by means of a communications line.
- 5 In general, an adjective describing an item or operation that is close at hand or restricted to a particular area.

local class

- 1 A class declared inside a block.
- 2 Or application local class. In Windows, any window class that an application registers for its exclusive use. Although an application can register any number of local classes, most applications register only one. This window class supports the window procedure of the application's main window. Windows destroys a local class when the application that registered it closes.

local machine

Or local computer. A computer that is accessed directly by the user rather than through an intermediate computer using a communications device. The opposite of a remote machine.

local object

- 1 In C++, an object created inside a function or smaller enclosing block. A local object has scope only within the block in which it is created.
- 2 In OLE, an object (that is not an embedded object) inside a container. Sometimes referred to as a cross-process object.

local scope

In C++, the degree of visibility afforded to a name (a variable, for example) when it is declared within a block of code. The name is accessible only from the point of declaration to the end of the block in which it is declared. See also class scope, file scope, function scope, function-prototype scope.

local variable

Or automatic variable. A variable declared within a function body, or smaller enclosing block, and not declared as static, that exists only within the scope of the enclosing curly braces ({ }). If a local variable has an initializer, it is initialized every time it is created and its contents are undefined when the function returns.

locale

The national and cultural environment in which a system or program is running. The locale determines the language used for messages and menus, the sorting order of strings, the keyboard layout, and date and time formatting conventions. See also code page, locale identifier (LCID).

locale identifier

A 32-bit value identifying the language and sublanguage for a locale. The locale identifier is used to customize string handling. See also code page, language identifier (ID).

locale-specific

Behavior that depends on local conventions of nationality, culture, and language, such as language, money format, and date format.

locking mode

A strategy for locking records in a recordset during update. A record is locked when it is read-only to all users but the one currently entering data in it. See also optimistic locking, pessimistic locking.

logical brush

An ideal description of a brush bitmap. A logical brush describes all the attributes (style, color, and so on) specified by the application that created it, although some may not be representable on available output devices. Windows also provides seven predefined logical stock brushes. See also brush, physical brush.

logical color palette

An array of colors created by an application that can be selected into a device context and used for graphics output. A logical color is an individual color in the **PALETTE array**. An application using a logical color palette can pass a **COLORREF** value, instead of an explicit red, green, blue (RGB) value, to GDI functions that expect a color. See also color palette, device context (DC).

logical coordinate system

A conceptual coordinate system that a program uses when giving drawing commands. Windows converts logical coordinates to the output device's physical coordinates when it renders the image.

logical font

An ideal description of a font. The font itself is described by the attributes (height, width, orientation, and so on) defined for it by an application. Windows also provides six predefined logical stock fonts. Before an application can begin drawing text with a logical font, it must find the closest match from among the physical fonts stored on the device or in the operating system. See also font mapper, physical font.

logical operator

An operator that performs Boolean evaluations on its operands. The logical-NOT (!) operator produces the value 0 if its operand is true (nonzero) and the value 1 if its operand is false (0). The logical-AND (&&) operator produces the value 1 if both operands have nonzero values; otherwise, it produces the value 0. The logical-OR (||) operator produces the value 1 if either of its operands has a nonzero value. See also bitwise operator.

logical shift

Or bitwise shift. A bitwise operation wherein the bits in the first operand are shifted by the number of positions specified by the second operand. The operator specifies the direction the bits are shifted. See also arithmetic shift.

logical unit

A conceptual unit of measure for graphics device interface (GDI) functions. A logical unit is converted to a device, or physical, unit (for example, to a number of pixels or to a distance in inches) when an image is rendered on the output device. By default, GDI considers logical units to be equal to device units, meaning that 1 logical unit equals 1 pixel on the screen.

logical view

A view expressed in terms of logical coordinates. A logical view can change its scaling when it is rendered onto an output device such as a display monitor.

lookup table

A reference table that maps an index or key to a value to be looked up and returned. A lookup table is often used as an alternative to lengthy run-time calculations.

loop

1 A set of statements in a program executed repeatedly, either a fixed number of times or until a condition is true or false.

2 In an event-driven state machine, an action that returns control to the original state, either directly or through a series of state transitions.

loss of significance

In programming, a potential error caused by presenting an argument to a math function that is either so large or so small that the operation returns a value that has lost most or all of the significant digits of the original argument. See also underflow.

low order

A description applied to the rightmost element in a group—the one that carries the least weight or significance. For example, the low-order bit in a byte would be the rightmost one.

LRPC

In OLE, a protocol for interprocess communication on a single machine. See also Remote Procedure Call (RPC).

luminance

In graphics, the perceived brightness of a surface or pixel. Luminance often refers to as a weighted average of red, green, and blue color values that gives the perceived brightness of the combination. See also red, green, blue (RGB); red, green, blue, alpha (RGBA).

Glossary M

machine code

The ultimate result of the compilation of assembly language or any high-level language, such as C++ sequences of bytes that are loaded and executed by a microprocessor. Machine code is the only language that computers understand; all other programming languages represent ways of structuring human language so that humans can get computers to perform specific tasks.

Macintosh binary resource file

A Macintosh resource file that has been created from a Windows resource script (.RC) and compiled using the Windows Portability Library version of the Windows Resource Compiler (RC.EXE). By default, .RSC is the filename extension for Macintosh binary resource files.

Macintosh Resource Compiler

A command-line tool, MRC.EXE, controlled by a .R resource script file, that compiles an application's Macintosh-specific resources.

macro

- 1 In C and C++, a name defined by the **#define** directive that specifies replacement text for subsequent invocations of the macro in the translation unit.
- 2 In applications, a set of keystrokes and instructions recorded and saved under a short key code or macro name. When the key code is typed or the macro name is used, the program carries out the instructions of the macro.

macro definition

- 1 In a C/C++ source file, the **#define** preprocessor directive, followed by an identifier, followed by an expression or statement. During compilation, every occurrence of the identifier as a token (that is, not in a comment or a literal string) is replaced with the expression.
- 2 In general, a name and a set of instructions that are recorded in a program so that the instructions can be substituted for the name.

macro expansion

In C/C++, the process of replacing a macro name (defined with a **#define** directive) in the source code with the body of the macro during compilation.

main application window

Or main window. In an SDI application, the window that serves as the primary interface between the user and the application.

main frame window

The primary window responsible for coordinating the frame with its view. In an SDI application, the document frame window is also the main frame window. In an MDI application, document windows are child windows displayed in the main frame window. The styles and other characteristics of frame windows are inherited from the main frame-window class.

main message

The main loop in an application that retrieves messages from a message queue and dispatches them to the appropriate window procedures. The main message loop takes over when there is no secondary loop to handle a particular message. Every application for Windows has a main message. See also message queue.

major version number

In a software version number, the number on the left of the decimal point. For example, the major version number of Windows NT version 3.51 is 3. See also minor version number.

makefile

A file that contains all commands, macro definitions, options, and so on to specify how to build the projects in a project workspace. A makefile has the filename extension .MAK and usually has the same base name as the workspace configuration (.MDP) file.

manifest constant

Or symbolic constant. In C/C++, an identifier defined in a #define preprocessor directive to represent a constant value. For example, `#define PI 3.14` declares a manifest constant named `PI`. In C++, the keyword **const** can be used in place of the #define directive for declaring a constant.

mapfile

A text file that contains information about the program being linked, including the groups in the program and a list of public symbols. The linker names the mapfile with the base name of the program and the filename extension .MAP.

MAPI

A set of functions that is part of the Win32 API that applications use to create, manipulate, transfer, and store various kinds of communications, such as e-mail messages. It provides application development tools for defining message purpose and content, and allows flexible management of stored messages. MAPI also provides a common development interface for creating communication-enabled and communication-aware applications independent of the underlying messaging system.

marshaling

In OLE, the process of packaging and sending interface parameters across process boundaries. See also Remote Procedure Call (RPC).

mask

- 1 A binary value used to selectively screen out or let through certain bits in a data value.
- 2 (verb) An operation used to select certain bits from a data value, by using a logical operator to combine the mask and the data value. For example, the mask 0x3F, when used with the logical-AND **&&** operator, removes (masks off) the two uppermost bits in an 8-bit data value but does not affect the rest of the value.

mask bitmap

A bitmap that selectively lets through or screens out values in another bitmap. This is accomplished by combining the two bitmaps in a bitwise logical operation.

maximized window

An application window enlarged to fill the entire desktop, a document window enlarged to fill the entire application workspace, or a client window enlarged to fill the client area of the client window. See also minimized window.

MBCS

A character set in which each character is represented as either a 1-byte or 2-byte value. The standard ASCII character set is a subset of the MBCS, which is used for languages that require a set of more than 256 characters. See also double-byte character set (DBCS), lead byte, trail byte, Unicode.

MDI

The standard user-interface architecture for Windows-based applications. A multiple document interface application enables the user to work with more than one document at the same time. Each document is displayed within the client area of the application's main window. See also child window, client area, single document interface (SDI).

member

In C/ C++, one of the elements of a structure, union, or (in C++) a class. In a class, both variables and functions are considered members of the class.

member function

In C++, a function declared inside a class definition. A class's member functions are used to get and set data members, display information to the user, and manipulate data according to the needs of the program. See also data member.

member-access control

Part of the functionality of C++ that prevents data from being used in ways that the programmer did not intend. Member-access control is implemented by specifying class members as private, protected, or public.

memory allocation

A response by the operating system to a request from a program for memory. The two basic types of memory allocation are static allocation, in which memory is set aside when the program starts and remains allocated while the program is running, and dynamic allocation, in which memory is allocated and deallocated while the program is running.

memory block

A section of random-access memory temporarily assigned to a program by the operating system.

memory device context

A block of memory that represents a display surface. A memory device context can be used to prepare images in memory before copying them to the actual display surface of the compatible device.

memory leak

A loss of memory resources that occurs when memory is allocated on the heap and never deallocated to make it available for reuse.

memory model

The approach used to address the code and data used in a computer program. The memory model dictates how much memory can be used in a program for code and how much for data. Most computers with a flat address space support only a single memory model. Computers with a segmented address space usually support multiple memory models. For example, in the 16-bit Microsoft C small memory model, code, data, and data arrays can each be a maximum of 64K in total memory.

memory space

See address space.

menu resource

A collection of information that defines the appearance and function of an application menu. This information includes the text strings that appear in an application's menu bar, menu-item identifiers, the arrangement of the menus and status of menu items.

menu-item identifier

Or menu-item ID. The command identifier chosen for a menu command during the creation of a menu item. Menu-item identifiers can be preassigned values (for example, assigned by the Microsoft Foundation Class Library) or values specified by the programmer. `ID_EDIT_CLEAR_ALL` is the preassigned menu-item ID for a Clear All command on the Edit menu.

message

A structure or set of parameters used for communicating information or a request. Messages can be passed between the operating system and an application, different applications, threads within an application, and windows within an application.

message box

A window that displays information to the user. For example, a message box can inform the user of a problem that the application has encountered while carrying out a task.

message filter

A filter that an application uses to retrieve specific messages from the message queue (while ignoring other messages). The filter is a range of message identifiers (specified by a first and last identifier), a window handle, or both.

message handler

In Windows, an object, such as a view or frame window, that provides handler functions to process messages.

message handling

The act of responding to messages received from the operating system. Applications use message-handling functions to process messages.

message loop

A program loop that retrieves messages from a thread's message queue and dispatches them to the appropriate window procedures. See also message queue.

message map

A mechanism to route Windows messages and commands to the windows, documents, views, and other objects in an MFC application. Message maps map Windows messages, commands from menus, toolbar buttons, accelerators and control-notification messages. A collection of macros in a class's source (.CPP) file specify which functions will handle various messages for the class.

message pump

A program loop that retrieves messages from a thread's message queue, translates them, offers them to the dialog manager, informs the MDI manager about them, and dispatches them to the application. See also message queue.

message queue

A repository for window messages awaiting processing by a thread. The system message queue holds mouse and keyboard input waiting to be passed to a thread's message queue. A thread's message queue holds messages waiting to be retrieved by a thread's message loop.

message-driven

Or event-driven. A programming model in which the state of the running program changes in response to user actions and other events. These events cause the operating system to send messages to the part of the application that can handle the event.

message-handler function

In Windows, a function that responds to a message using parameters that have been translated from *wParam* and *lParam* and passed to the function. A message-handler function might track mouse activity or call member functions of the document to update its data.

message-map entry

An individual item in a message-map table that specifies the handler for a particular message. The framework searches the table for each incoming message and calls the handler for it automatically. See also message map.

message-map macro

One of the macros supplied by the Microsoft Foundation Class Library and used in a message map to specify which messages will be handled by which functions. Message-map macros are available to map Windows messages, command messages, and ranges of messages.

Messaging Application Programming Interface

A set of functions that is part of the Win32 API that applications use to create, manipulate, transfer, and store various kinds of communications, such as e-mail messages. It provides application development tools for defining message purpose and content, and allows flexible management of stored messages. MAPI also provides a common development interface for creating communication-enabled and communication-aware applications independent of the underlying messaging system.

metafile

A collection of structures that stores a picture in a device-independent format. A metafile defines an image as coded lines and shapes. Win32 uses enhanced-format metafiles, which have a .EMF filename extension. Windows metafiles, which have a .WMF filename extension, are used in

Windows 3.x.

method

In object-oriented programming, a procedure that provides access to an object's data. In C++, public member functions are the equivalent of methods.

metric

- 1 A dimension (width or height) of a display element, such as a cursor.
- 2 One of the attributes of a font as described in a font specification. For example, the character placement and line-spacing metrics enable an application to compute device-independent line breaks that are portable across screens, printers, typesetters, and even platforms.

MFC

A set of C++ classes that encapsulate much of the functionality of applications written for the Microsoft Windows operating systems.

Microsoft Foundation Class Library

A set of C++ classes that encapsulate much of the functionality of applications written for the Microsoft Windows operating systems.

Microsoft Internet Information Server

A Web server integrated into Windows NT Server which provides FTP, HTTP, and gopher services.

MIIS

A Web server integrated into Windows NT Server which provides FTP, HTTP, and gopher services.

MIME

A standard that allows binary data to be published and read on the Internet. The header of a file with binary data contains the MIME type of the data; this informs client programs (Web browsers and mail packages, for instance) that they will need to handle the data some way other than they handle straight text. For example, the header of a Web document containing a JPEG graphic contains the MIME type specific to the JPEG file format. This allows a browser to display the file with its JPEG viewer, if one is present.

mini-server application

A type of server that can only be launched from a container and that only supports embedding, not linking. Objects created by mini-servers are always stored within compound documents, never as individual files. Microsoft Draw is an example of a mini-server. See also full-server application.

minimal rebuild

A feature that permits the build engine to analyze dependencies within your project and to skip compilation of files that it detects are unaffected by a change, even if those files include changed headers.

minimized window

A window that has been shrunk to an icon. See also maximized window.

minor version number

In a software version number, the number on the right of the decimal point. For example, the minor version number of Microsoft Windows NT version 3.51 is 51. See also major version number.

mnemonic key

See access key.

mnemonics

In a graphical user interface, underlined letters in the text of menu and dialog-box items. When the menu and dialog-box items are active, the user can select the item by pressing the key that corresponds to the underlined letter. See also access key.

modal

A restrictive or limiting interaction created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to its primary window or to the entire system. A modal dialog box must be closed by the user before the application continues. See also modeless, system modal dialog box.

modal loop

A message-processing loop during which the system retrieves and dispatches messages without allowing an application the opportunity to filter the messages in its main message

modeless

Not restrictive or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with other windows. A modeless dialog box stays on the screen and is available for use at any time but also permits other user activities. See also modal.

modifier

1 In C and C++, a reserved word in a declaration that qualifies the identifier. The storage-class specifier **extern** and the calling-convention keyword **__declspec(thread)** are examples of modifiers.

2 Generally, anything that limits or restricts meaning, as in a modifier flag.

modifier key

A keyboard key that changes the action of normal input when it is pressed. For example, CTRL, ALT, and SHIFT are modifier keys.

module-definition file

A text file that contains one or more statements describing various attributes of an executable module. Module-definition files usually have a .DEF filename extension. See also dynamic-link library file.

moniker

An object that supports the **IMoniker** interface, which includes the **IPersistStream** interface; thus, monikers can be saved to and loaded from streams. Monikers can be saved to persistent storage, and they support binding.

Mosaic

A World Wide Web browser written by NCSA.

most recently used

A file list, control of which is supported by the **CRecentFileList** class in MFC. Files can be added to or deleted from the MRU file list. The file list can be read from or written to the registry or an .INI file, and the menu displaying the MRU file list is updatable.

mount

To make a volume (a physical disk) accessible to a computer's file system. The volume could be a local drive or one accessed through a network connection. The term usually describes accessing disks in Apple Macintosh and UNIX-based computers.

mouse capture

The act of channeling mouse input to a specific window without regard to the position of the mouse-cursor hot spot.

mouse event

An input event that occurs whenever the user moves the mouse, or presses or releases a mouse button. Windows converts mouse input events into messages and posts them to the appropriate thread's message queue. See also mouse message.

mouse message

A message sent to a window when a mouse event occurs while the cursor is within the borders of the window, or when the window has captured the mouse. Mouse messages are divided into two groups: client-area messages and nonclient-area messages. See also mouse capture.

Moving Picture Experts Group

Compression algorithms for both video and sound. There are multiple standard levels for MPEG (MPEG1, MPEG2, and so on). Movies saved in the MPEG format are highly compressed and consume much less disk space than would normally be required.

MP

The Internet Engineering Task Force (IETF) standard for aggregating multiple ISDN B channels using synchronous PPP framing.

MPEG

Compression algorithms for both video and sound. There are multiple standard levels for MPEG (MPEG1, MPEG2, and so on). Movies saved in the MPEG format are highly compressed and consume much less disk space than would normally be required.

MRC

A command-line tool, MRC.EXE, controlled by a .R resource script file, that compiles an application's Macintosh-specific resources.

MRU

A file list, control of which is supported by the **CRecentFileList** class in MFC. Files can be added to or deleted from the MRU file list. The file list can be read from or written to the registry or an .INI file,

and the menu displaying the MRU file list is updatable.

multibyte character set

A character set in which each character is represented as either a 1-byte or 2-byte value. The standard ASCII character set is a subset of the MBCS, which is used for languages that require a set of more than 256 characters. See also double-byte character set (DBCS), lead byte, trail byte, Unicode.

multiline edit control

A text box that displays more than one line of text. In a multiline edit control, data that is too long to fit on one line can either wrap to the next line or extend beyond the right boundary of the box. See also single-line edit control.

Multilink PPP

The Internet Engineering Task Force (IETF) standard for aggregating multiple ISDN B channels using synchronous PPP framing.

multiple document interface

The standard user-interface architecture for Windows-based applications. A multiple document interface application enables the user to work with more than one document at the same time. Each document is displayed within the client area of the application's main window. See also child window, client area, single document interface (SDI).

multiple document types

More than one kind of document—for example, a text document and a graphics document. In an application (such as Microsoft Excel) that supports multiple document types, each document type puts its own menus into the menu bar and changes the main frame window's caption.

multiple-selection list box

A list box in which more than one item can be selected at the same time.

Multipurpose Internet Mail Extensions

A standard that allows binary data to be published and read on the Internet. The header of a file with binary data contains the MIME type of the data; this informs client programs (Web browsers and mail packages, for instance) that they will need to handle the data some way other than they handle straight text. For example, the header of a Web document containing a JPEG graphic contains the MIME type specific to the JPEG file format. This allows a browser to display the file with its JPEG viewer, if one is present.

multithreaded application

An application capable of carrying out multiple, independent paths of execution (or threads) at the same time. For example, an application may have additional threads to handle background or maintenance tasks so that the user doesn't have to wait for a task to complete before continuing to work with the application. See also user-interface thread, worker thread.

mutex object

In interprocess communication, a synchronization object whose state is signaled when it is not

owned by a thread and nonsignaled when it is owned. Only one thread at a time can own a mutex.

Glossary N

name decoration

The process during compilation of a C++ function definition or prototype in which the compiler creates a string that uniquely identifies a function. Name decoration is necessary for the compiler and linker to distinguish between several functions that may have the same name. See also decorated name.

National Center for Supercomputer Applications

A research center at the University of Illinois in Urbana-Champaign for developing and implementing a strategy to create, use, and transfer advanced computing and communication tools and information technologies. NCSA developed Mosaic.

native

- 1 An adjective describing a programming element that is specific to and supported by the particular platform on which it is running.
- 2 An adjective describing a programming element that is specific to a particular country.
- 3 In OLE, an adjective describing data that originated in the container document.

NCSA

A research center at the University of Illinois in Urbana-Champaign for developing and implementing a strategy to create, use, and transfer advanced computing and communication tools and information technologies. NCSA developed Mosaic.

nest

To embed one construct (for example, a function) completely within another.

nested class

A class that is declared within the scope of another class. A nested class is available for use within the scope of the enclosing class. To refer to a nested class from a scope other than its immediate enclosing scope, a fully qualified name must be used.

Network Service Provider

A company that provides connectivity to the Internet for ISPs and others requiring high speed connections between their LANs and the Internet.

New Technology file system

A fixed-disk file system that offers security features, including execute-only files and permissions for individual files, support for Unicode and long filenames, and the ability to handle large storage media. See also high-performance file system (HPFS).

newline character (\n)

The character used to mark the end of a line in a text file, or the escape sequence (\n) used to represent this character.

NMAKE

The Microsoft Program Maintenance Utility. NMAKE reads a description file that specifies project-file

dependencies and automatically executes the commands needed to update the project when any project file has changed.

nonclient area

The parts of a window that an application does not use when displaying output such as text or graphics. A window's nonclient area consists of the border, menu bar, title bar, scroll bar, Control menu, Minimize button, and Maximize button. See also client area.

nonintegral expression

An expression that evaluates to a data type other than a **char**, **short**, **int**, **long**, or enumerated type.

nonscalar type

See aggregate type.

nonscalar value

A value whose type is a complex data structure such as a structure, class, or array. See also aggregate type.

nonsystem key

A keyboard key that is pressed when the ALT key is not pressed or a keyboard key that is pressed when a window has the keyboard focus.

notification message

A message that a control sends to its parent window when events, such as input from the user, occur.

NSP

A company that provides connectivity to the Internet for ISPs and others requiring high speed connections between their LANs and the Internet.

NT-1

In ISDN, a network terminator composed of a connector that attaches a two-wire ISDN line to a four-wire line so that it can be connected to PCs and terminals.

NTFS

A fixed-disk file system that offers security features, including execute-only files and permissions for individual files, support for Unicode and long filenames, and the ability to handle large storage media. See also high-performance file system (HPFS).

null brush

Or hollow brush. A logical brush created from a bitmap whose color matches the current window background color. See also bitmap.

null pointer

A pointer to nothing, expressed in C++ as the value 0.

null statement

A statement that performs no action, declares nothing, and can be used wherever a statement is

expected. In C/C++, a statement that contains only a semicolon (;).

null string

Or empty string, zero-length string. In C/C++, a string that contains no characters. A null string is explicitly initialized with a pair of double quotation marks ().

null terminator

Or string terminator. In C/C++, the null character '\0' at the end of a string.

null-terminated character string

Or ASCIIZ string. A string of characters terminated by a single character, '\0', whose integer value is 0.

Glossary O

object

Generally, an instance of an entity that embodies both specific data and the functions that manipulate it. Specifically in object-oriented programming, an object is an entity that has state, behavior and identity. An object's state consists of its attributes and the attributes' current values. An object's behavior consists of the operations that can be performed on it and the accompanying state changes. An object's identity is what you use to distinguish it from other objects. In contrast, COM objects' behavior is defined by the interfaces it supports. A COM object's state is not explicitly specified, but is implied by its interfaces. A COM object's identity is defined by the ability to use **Unknown::QueryInterface** to move between interfaces.

object code

Executable code generated by a compiler or an assembler from the source code of a program. See also object file.

object description language file

In OLE Automation, text files containing a description of an application's interface. Object description language scripts are compiled into type libraries using the MkTypLib tool included with the OLE Software Development Kit.

object file

A file containing object code and/or data generated by a compiler or an assembler from the source code of a program. Object files generated by the Visual C++ compiler have a .OBJ filename extension. See also Common Object File Format (COFF).

object library file

A dynamic-link library with a type library resource. An object library file typically has a .OLB filename extension.

Object Linking and Embedding

Pronounced o-LAY. A way to transfer and share information between applications. Linking and embedding are two methods in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file.

Object Module Format

A specification for the structure of object (.OBJ) files. The Microsoft Visual C++ version 4.0 linker accepts object files that are either COFF or 32-bit OMF. See also Common Object File Format (COFF), EDITBIN.

object-oriented design

Or object-oriented programming. In traditional procedural languages (such as C, Fortran, and Cobol) code and data are separate. In the object-oriented approach, code and data that belong together can be combined into objects. Object-oriented design is further characterized by the use of

inheritance (derived classes), polymorphism, encapsulation, and virtual functions (C++) in programming.

ODBC

An open, vendor-neutral interface for database connectivity that provides access to a variety of personal computer, minicomputer, and mainframe systems, including Windows-based systems and the Apple Macintosh. The ODBC interface permits an application developer to develop, compile, and ship an application without targeting a specific database management system (DBMS). Users can add modules called database drivers that link the application to their choice of database management systems.

ODBC cursor library

A dynamic-link library that resides between the ODBC Driver Manager and the drivers and handles scrolling through data.

ODBC driver

A dynamic-link library file that implements ODBC function calls and interacts with a data source. A driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated database management system.

OEM character set

Printable characters used in full-screen MS-DOS sessions for screen display. Characters 32 through 127 are usually the same in the OEM, U.S. ASCII, and Windows character sets. The other characters in the OEM character set (0 through 31 and 128 through 255) correspond to the characters that can be displayed in a full-screen MS-DOS session. These characters are generally different from the Windows characters and may vary from one original equipment manufacturer (OEM) to another.

offset

In relative addressing methods, a number that tells how far from a starting point a particular item is located. For example, in the search for a specific data item stored within a known area (segment) of memory, an offset is used to tell the microprocessor how many bytes past the beginning of the segment the item is located.

OLE

Pronounced o-LAY. A way to transfer and share information between applications. Linking and embedding are two methods in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file.

OLE Automation

A way to manipulate an application's objects from outside the application. OLE Automation is typically used to create applications that expose objects to programming tools and macro languages, to create and manipulate one application's objects from another application, or to create tools for accessing and manipulating objects.

OLE Automation object

Or exposed object. An instance of a class that is defined within an application, exposed for

access by other applications or programming tools by means of OLE Automation interfaces. See also OLE Automation client, OLE Automation server.

OLE Automation server

An application that exposes programmable objects to other applications, which are called automation clients. Exposing programmable objects enables clients to automate certain functions by directly accessing those objects and using the services they make available. For example, a word processor might expose its spell-checking functionality so that other programs can use it. See also automation client.

OLE item

Or OLE (pronounced o-LAY) object. An object that represents data, created and maintained by a server application, that can be seamlessly incorporated into a document so that the object appears to be a part of the larger document. The result is a compound document made up of the OLE item and a containing document.

OLE system DLLs

Pronounced o-LAY. The means by which OLE containers and OLE servers communicate. The OLE system dynamic-link libraries (DLLs) provide functions that containers and servers call, and the containers and servers provide callback functions that the DLLs call. Using this means of communication, a container doesn't need to know the implementation details of the server application. A container can accept items created by any server without having to define the types of servers with which it can work

OMF

A specification for the structure of object (.OBJ) files. The Microsoft Visual C++ version 4.0 linker accepts object files that are either COFF or 32-bit OMF. See also Common Object File Format (COFF), EDITBIN.

Open Database Connectivity

An open, vendor-neutral interface for database connectivity that provides access to a variety of personal computer, minicomputer, and mainframe systems, including Windows-based systems and the Apple Macintosh. The ODBC interface permits an application developer to develop, compile, and ship an application without targeting a specific database management system (DBMS). Users can add modules called database drivers that link the application to their choice of database management systems.

Open System Interconnection

A model that breaks network management into five functional areas: fault management, configuration management, security management, performance management, and accounting management.

operand

An entity on which an operator acts.

operator

Symbols that specify an evaluation to be performed on one operand (unary operator), two operands (binary operator), or three operands (ternary operator). See also operator precedence.

operator precedence

The relative position of an operator in the hierarchy that determines the order in which expressions are evaluated.

optimistic locking

A recordset locking strategy in which records are left unlocked until explicitly updated. The page containing a record is locked only while the program updates the record, not while a user is editing a record. See also pessimistic locking.

optimization

Compiler fine tuning to increase program performance or reduce program size. Visual C++ provides five optimization options: Default, Disable (Debug), Maximize Speed, Minimize Size, and Customize.

OSI

A model that breaks network management into five functional areas: fault management, configuration management, security management, performance management, and accounting management.

out-of-band data

A logically independent transmission channel associated with each pair of connected stream sockets. Out-of-band data is delivered to the user independently of normal data. The abstraction defines that the out-of-band data facilities must support the reliable delivery of at least one out-of-band message at a time. This message may contain at least 1 byte of data, and at least one message may be pending delivery to the user at any one time.

out-of-memory exception

An error that occurs during the execution of a program when an out-of-memory situation is encountered.

output precision

In specifying a font, defines how closely the output must match the requested font's height, width, character orientation, escapement, and pitch.

outside-in

A model for in-place OLE objects that requires the user to double-click or select a verb from the Edit menu to activate the object. Outside-in objects are hidden from the user when inactive. See also inside-out.

overhang

The amount by which the black part of a glyph extends beyond its character cell on either side. See also ABC width, underhang.

overlapped I/O

See asynchronous operation.

overlapped window

A style of window meant to serve as an application's main window. Other windows can overlap the window's space on the screen.

overloaded operator

In C++, an operator that has been given functionality beyond that which is built into the language. For example, the programmer can overload the arithmetic operators to allow them to use objects of a given class as their operands. Overloading is usually done to make the source code more readable.

overloading

Supplying more than one definition for a given function name or operator name in the same scope. For example, the addition (+) operator can be overloaded to add the values contained in two objects of the same class. See also function overloading.

owned window

A window that has an owner. An owned window always appears in front of its owner window, is hidden when its owner window is minimized, and is destroyed when its owner window is destroyed. See also owner window.

owner window

A window that owns another window, thus affecting aspects of the owned window's appearance and behavior. See also owned window.

owner-draw control

In Windows, a control that is customized for a specific application. Owner-draw controls are similar to predefined controls in that Windows will handle the control's functionality and process input from the mouse and keyboard. However, the programmer is responsible for the appearance of the owner-draw control in its various states.

Glossary P

package

- 1 In VC++ Developer Studio, a dynamic-link library that extends the development environment.
- 2 In Java, a group of classes declared with the **package** keyword. Classes are generally packaged together when they are related in a significant way. In Java, the default level of data access is at the package level. In C++, the default level of data access is at the class level.

packaged function

A function created by the compiler when function-level linking is enabled. A packaged function is visible to the linker in the form of a COMDAT record placed in the object (.OBJ) file by the compiler. Functions that are not packaged can be linked only at the object level.

Packet Internet Groper

A program useful in testing and debugging LAN/WAN troubles. It sends out a packet and expects a specified host to respond back in a specified time frame.

palette

- 1 An array of colors currently available on a device.
- 2 A modeless secondary window that displays a toolbar or another set of controls.

pane

One of the separate areas in a split window, or a rectangular area of the status bar that can be used to display information.

PAP

In ISDN, a type of authentication accomplished by sending user ID/password pairs between two devices.

parameter

Or argument. A value or an expression used with an operator or passed to a subprogram (subroutine, procedure, or function). The program then carries out operations using the arguments. Parameters are also used with templates; such parameters can include classes as well as values and expressions.

parameter data member

In recordsets, a data member that accepts parameter information at run time. Parameter data members can be used in a query to select records from a database based on the value of the parameter obtained or calculated at run time. See also field data member.

parameter passing

The sending of variables as parameters (arguments) to a subroutine, procedure, or function. You can pass parameters by value or by reference. Passing by value copies the variable's value into the subroutine's formal argument. Passing by reference copies the variable's memory address into the subroutine's formal argument. In C++, parameters may be passed by value or by reference. In C, all parameters are passed by value, except for arrays, which are translated into the address of the

first member and reference of type, as in `int&ri`. However, you can force a pass by reference by using pointers or references. In Java, parameter passing is by value only for simple types, by reference only for instances.

parent process

In a multitasking environment, a process that initiates one or more processes, called child processes. The parent process defines the environment for child processes. The parent process can suspend or complete its own processing while the child processes continue operating.

parent window

A window that generates one or more child windows.

parse map

Code that allows a **CHttpServer**-derived object in MFC to map client requests to its member functions. MFC provides parse map macros to help map form variables on a Web page to parameters in your function.

parser

The part of a compiler, interpreter, or other translator that accepts tokens from the lexical analyzer and attaches semantics to those tokens.

Pascal calling convention

A calling convention that pushes arguments onto the stack in the order in which they appear in the function call (from left to right). The Pascal calling convention requires that the called routine remove arguments from the stack.

Password Authentication Protocol

In ISDN, a type of authentication accomplished by sending user ID/password pairs between two devices.

path bracket

One or more graphics device interface (GDI) functions embedded between a **BeginPath** and an **EndPath** function. (The **BeginPath** function defines the start of the bracket; the **EndPath** function defines the end.)

pen

In graphics applications, a drawing object used to draw lines and borders.

PERL

A general-purpose scripting language used mainly for server-side HTML programming. It is freely available on a wide variety of platforms.

permanent storage

- 1 A disk medium, such as a hard disk or floppy disk.
- 2 A place to store data, such as a file or database.
- 3 In the Visual C++ documentation, a variable declared with the **static** storage-class specifier.

persistent

Lasting between program sessions, or renewed when a new program session is begun.

pessimistic locking

A recordset locking strategy in which a page is locked once a user begins editing a record on that page. While the page is locked, no other user can change a record on that page. The page remains locked until records are updated or the editing is canceled. See also optimistic locking.

physical brush

A bitmap that Windows uses to paint the interior of filled shapes. A physical brush is a device driver's approximation of a logical brush.

physical device

1 The fourth and final coordinate space for most graphics device interface (GDI) drawing operations. (The Win32 API uses four coordinate spaces: world, page, device, and physical device.) The points in this physical device coordinate space can correspond to the desktop, a client area, or a page of printer paper.

2 Hardware such as a modem or printer to which data can be written.

physical font

A font stored internally on a device or whose resources have been loaded into the operating system. See also logical font.

PING

A program useful in testing and debugging LAN/WAN troubles. It sends out a packet and expects a specified host to respond back in a specified time frame.

pipe

A communication conduit with two ends. A process with a handle to one end can communicate through a pipe with a process having a handle to the other end. Pipes can be one way (where one end is read-only and the other end is write-only) or two way (where both ends of the pipe can be used for reading or writing). The Win32 application programming interface (API) provides both anonymous (unnamed) pipes and named pipes.

pixel

The smallest addressable picture element (that is, a single dot) on a display screen or printed page. See also raster.

placeholder

A symbol in an expression indicating information to be supplied. For example, in the following SQL statement

```
SELECT CourseID, CourseTitle FROM Course WHERE CourseID = ?
```

the question mark (?) is a placeholder for a parameter value supplied at run time.

platform

The hardware and operating system that support an application. Platform sometimes refers to the hardware alone, as in the Intel x86 platform.

point

In typography, a unit of measurement for type (1 point equals approximately 1/72 inch).

Point-to-Point Protocol

An advanced serial packet protocol, similar to but an improvement on SLIP, commonly used for dial-up connections. PPP is an official Internet protocol; SLIP is not.

pointer

1 A data type that can contain the address of another data type such as a variable, or of a function or class.

2 A graphic image displayed on the screen that indicates the location of a pointing device (sometimes called a cursor).

polygon-filling mode

In graphics, an algorithm that determines the parts of a region that can be filled, painted, inverted, or clipped.

polymorphism

In C++, the concept of a single interface for multiple functions. For example, different classes can each contain a function called `print` that prints data in a format appropriate for objects of that class. The compiler selects the appropriate print function for each call to `print`. See also virtual function.

pop-up menu

A menu that is hidden until the user performs an action (such as clicking the right mouse button) that causes Windows to display the menu. The pop-up menu contains commands that are relevant to the selection or the active window. See also drop-down menu.

pop-up window

An immovable, nonsizable window that remains on the screen until the user dismisses it. Pop-up windows typically contain definitions of terms or other parenthetical information. Good uses for pop-up windows include illustrations, examples, notes, tips, and lists of keyboard shortcuts.

portable code

Code that can be compiled with little or no change to run on more than one platform.

Portable Operating System for Unix

An IEEE standard that defines the language interface between the UNIX operating system and application programs through a minimal set of supported functions.

POSIX

An IEEE standard that defines the language interface between the UNIX operating system and application programs through a minimal set of supported functions.

postfix notation

Placing of operators after operands, as in $xy+z^*$. Compare prefix notation, infix notation.

PPP

An advanced serial packet protocol, similar to but an improvement on SLIP, commonly used for dial-up connections. PPP is an official Internet protocol; SLIP is not.

Practical Extraction and Reporting Language

A general-purpose scripting language used mainly for server-side HTML programming. It is freely available on a wide variety of platforms.

pragma

An instruction to the compiler to perform a given action at compile time. For example, the pragma

#pragma pack([n])

specifies packing alignment for structure and union members.

precompiled header directive

A statement in a source or include file specifying where precompilation should stop.

precompiled header file

A file containing compiled code for a portion of a project. Subsequent builds combine this file with the uncompiled code, thus shortening the overall compile time. The default filename extension for a precompiled header file is .PCH.

predefined macro

A macro recognized by the compiler that takes no arguments and cannot be redefined. In most cases, the value of a predefined macro must be constant throughout compilation. For example, the ANSI predefined macro __TIME__ expands to the most recent compilation time of the current source file, expressed as a string literal of the form *hh:mm:ss*.

predefined query

Or stored procedure. In SQL, a set of one or more SQL statements stored in a data source, available to be called from an application as needed. Predefined queries reduce the overhead of repeatedly specifying the same selection criteria.

prefix character

In Windows text controls, the ampersand (&) character used to define the next character in the string as the mnemonic access key for the control. The ampersand is a directive to underscore the next character in the interface. A two-ampersand (&&) mnemonic-prefix character is a directive to print a single ampersand in the interface.

prefix notation

Placing of operators before operands, as in **+xyz*. Compare infix notation, postfix notation.

preprocessor

A text processor that manipulates the text of a source file as part of the first phase of translation. The preprocessor does not parse the source text, but it does break it up into tokens for the purpose of locating macro calls. Although the compiler ordinarily invokes the preprocessor in its first pass, the preprocessor can also be invoked separately to process text without compiling.

preprocessor directive

Or compiler directive. An instruction to the preprocessor to perform an action on source-program text before the compilation phase. For example, the directive `#include my_defs.h` instructs the preprocessor to include the contents of the file `my_defs.h` in the compilation unit.

presentation cache

In OLE, a container document's memory cache for the graphical representation of a linked or embedded object. Such a presentation is cached locally in the container document so it can be obtained without the expense of running the object's application. See also presentation data.

presentation data

In OLE, the graphical representation of a linked or embedded object.

primary key

In a database program, a field or group of fields that uniquely identify a record in a table. No two records in a table can have the same primary key value.

primary language

A language, such as English or Japanese, defined for a particular program. In some cases, a sublanguage is also specified. For example, English can be specified as the primary language and Canadian as the sublanguage. See also language identifier (ID).

primary thread

Or main thread. The initial thread of a process. When the primary thread terminates, so does the process. This primary thread is supplied to the operating system by the startup code in the form of a function address. Usually, it is the address of the **main** or **WinMain** function that is supplied.

primary token

An access token that is typically created only by the Windows NT executive layer. The primary token can be assigned to a process to represent the default security information for that process. See also access token, impersonation token, privilege, security identifier (SID).

primary verb

In OLE, the command that is executed when the user double-clicks an OLE item. For most types of OLE items, the primary verb is Edit, which launches the server that created the item. OLE items can support one or more verbs.

print processor

A dynamic-link library that receives information from the print spooler and sends the interpreted information to the graphics engine (another Windows DLL), which converts print processor output into device driver function calls. The printer processes these commands when it prints the image.

private

In C++, describes a limited degree of access to class members, as specified using the private keyword. Access to the specified class member is restricted to member functions and friends of the member or class.

privilege

In Windows NT, a locally unique identifier (LUID) used to control access to an object or service more strictly than is typical with discretionary access control. A system manager uses privileges to control which users are able to manipulate system resources. An application uses privileges when it changes a system-wide resource, such as when it changes the system time or shuts down the system. See also access token, impersonation token, primary token, security identifier (SID).

process

An executing application that consists of a private virtual address space, code, data, and other operating-system resources, such as files, pipes, and synchronization objects that are visible to the process. A process also contains one or more threads that run in the context of the process.

profiler

A development tool for analyzing the run-time behavior of programs. See also function profiling, line profiling.

profiler batch input file

In a profiling operation, a file that provides condensed information to the Visual C++ profiler (PROFILE). The PREP program generates a profiler batch input file the first time the profiler is run on a program. The default filename extension for profiler batch input files is .PBI. See also profiler batch output file, profiler batch text file.

profiler batch output file

An intermediate file generated by the Visual C++ profiler (PROFILE) and used to transfer information between profiling steps. See also profiler, profiler batch input file, profiler batch text file.

profiler batch text file

In a profiling operation, the file generated by the PREP program and used as input to the PLIST program to generate a human-readable profile of the source code. See also profiler, profiler batch input file, profiler batch output file.

program database file

A file used by the build tools to store information about a user's program. The program database file speeds linking during the debugging phase of development by keeping the debugging information separate from the object files.

progress bar control

In MFC, an application window that is used to indicate the progress of a lengthy operation. The control consists of a rectangle that is gradually filled, from left to right, with the system highlight color as an operation progresses.

prolog code

See prolog/epilog code sequence.

prolog/epilog code sequence

In Windows, a compiler-generated code sequence that executes on entry to a function. Prolog code allocates space on the stack, sets up addressing for the variables local to the function, and

initializes registers so that variables passed to the function can be conveniently accessed. The epilog code, which executes on exit from the functions, frees this space, restores the original contents of the registers, and ensures that any return value from the function is available to the caller.

property page

A grouping of properties presented as a tabbed page of a property sheet.

property set

In OLE structured storage, information describing a document, stored in a standard format so that other applications can locate and read that information. For example, a document created with a word processor can have a property set describing the author, title, and keywords.

property sheet

A special kind of dialog box that is generally used to modify the attributes of some external object, such as the current selection in a view. A property sheet has three main parts: the containing dialog box, one or more property pages shown one at a time, and a tab at the top of each page that the user clicks to select that page. An example of a property sheet is the Project Settings dialog box in the Microsoft Developer Studio.

protected

In C++, describes a limited degree of access to class members, as specified using the protected keyword. Access to the specified class member is restricted to member functions and friends of the class, as well as member functions of classes derived from the class.

protocol

1 In networking, a formal set of rules governing the format, timing, sequencing, and error control of exchanged messages on a data network; may also include facilities for managing a communications link and/or contention resolution; a protocol may be oriented toward data transfer over an interface, between two logical units directly connected, or on an end-to-end basis between two end users over a large and complex network

2 Rules and conventions of behavior for an action, such as argument passing, or an entity, such as the member functions of a class.

proxy server

Or firewall. A way to protect your local area network from being accessed by others on the Internet. The proxy server acts as a security barrier between your internal network and the Internet, keeping others on the Internet from accessing confidential information on your internal network.

pseudotarget

In a makefile, a label used in place of a filename in a dependency line. The Microsoft Program Maintenance Utility (NMAKE) interprets the label as a file that does not exist and executes the commands in the dependency line. The following is an example of a pseudotarget, `clean`, that instructs NMAKE to remove all of the object files and the executable file from the project directory:

```
clean:
    erase *.obj
    erase *.exe
```


public

In C++, describes a broad degree of access to class members, as specified using the public keyword. The specified class member can be accessed by any function. See also private, protected.

punctuators

In C++, characters (!, %, ^, for example) that have syntactic and semantic meaning to the compiler but do not, of themselves, specify an operation that yields a value. Some punctuators, either alone or in combination, can also be C++ operators or be significant to the preprocessor.

Glossary Q

qualified name

In C++, a name used with the binary scope-resolution (::) operator to disambiguate names that are reused within classes, structures, or unions. The name specified after the scope-resolution operator must be a member of the class specified on the left of the operator or a member of its base class(es).

query

A request for records from a data source. For example, a query can be written that requests, essentially, all invoices for Joe Smith, where all records in an invoice table with the customer name Joe Smith would be selected. See also recordset.

queue

A data structure in which elements are added to the end of a list and removed from the head of the list. A priority queue typically removes elements from the list according to some priority value assigned to each element. The system message queue, for example, holds mouse and keyboard input waiting to be processed.

Glossary R

r-value

The value on the right side of an expression. R-values are sometimes used to describe the value of the expression and to distinguish it from an l-value. An r-value can be one or more of the following: variable, constant, literal, mathematical expression, or function. See also l-value.

radio button

Or option button. In graphical user interfaces, a round button operated by the user to toggle an option or choose from a set of related but mutually exclusive options. The radio button has two states, selected (or checked) and cleared (or unchecked). When selected, a black dot appears inside the button's circle. See also radio group.

radio group

A number of radio buttons grouped together in a group-box control. A radio group allows the user to choose from a set of mutually exclusive options; selecting one radio button automatically clears the previously selected button.

radix

- 1 The number base for numeric input and displayed output. Visual C++ allows octal (8), hexadecimal (16), and decimal (10) radixes. The default radix is decimal.
- 2 In SQL, a column that identifies numeric data types; possible values are 10, 2, or NULL.

RAM

Semiconductor-based memory that can be read from and written to by the microprocessor or other hardware devices. The storage locations can be accessed in any order (i.e., randomly). While other types of memory, including read-only memory (ROM), can also be accessed randomly, RAM is generally understood to refer to volatile memory that can be written to as well as read.

random access memory

Semiconductor-based memory that can be read from and written to by the microprocessor or other hardware devices. The storage locations can be accessed in any order (i.e., randomly). While other types of memory, including read-only memory (ROM), can also be accessed randomly, RAM is generally understood to refer to volatile memory that can be written to as well as read.

RAS

A Windows NT Server feature by which a single serial connection provides a remote workstation with host connectivity, NT file services, or Novell file and printing services (NWLink).

raster

A horizontal line of pixels. A raster display device creates images composed of pixels, or dots. Computer monitors and laser printers are examples of raster devices. Devices such as plotters, which create images on paper by drawing lines with a pen, are not raster devices.

raster display

A video monitor that displays an image on the screen from top to bottom as a series of horizontal scan lines. The scan lines are as wide as the smallest visible image on the screen. Within each

scan line, individual picture elements, or pixels, can be illuminated. Television screens and most computer monitors are raster displays.

raster font

Or bitmap font, nonscalable font. A font in which each glyph (character or symbol) is of a particular size and style, designed for a specific resolution of device and described as a unique bitmap. See also scalable font, TrueType font, vector font.

raster operation

In computer graphics, a Boolean operation that combines either the values of the pixels in the selected pen and the destination bitmap (binary operation) or the values of the pixels in the source, the selected brush, and the destination (ternary operation). See also raster operation code.

raster operation code

A 32-bit code that defines how Windows combines colors in output operations that involve a current brush, a possible source bitmap, and a destination bitmap. See also raster operation (ROP).

rasterization

The conversion of vector graphics to equivalent images composed of pixel patterns that can be stored and manipulated as sets of bits. See also raster operation (ROP).

raw data

Unprocessed, typically unformatted data. Raw data is a stream of bits that has not been filtered for commands or special characters. More generally, it is information that has been collected but not evaluated.

read-only

Describes information stored in such a way that it can be played back (read) but cannot be changed (written). See also write-only.

record

1 (noun) A collection of data about a single entity, such as an account or a customer, stored in a row of a table. A record consists of a group of contiguous columns (sometimes called fields) that contain data of various types. See also recordset.

2 (verb) To retain information, usually in a file.

record field exchange

When bulk row fetching is not implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also bulk record field exchange (Bulk RFX), dialog data exchange (DDX), and DAO record field exchange (DFX).

record view

In form-based data-access applications, a form view object whose controls are mapped directly to the field data members of a recordset object and indirectly to the corresponding columns in a query result or table on the data source. See also form view, recordset.

recordset

A set of records selected from a data source. The records can be from a table, a query, or a stored procedure that accesses one or more tables. A recordset can join two or more tables from the same data source, but not from different data sources. See also dynaset, result set, snapshot.

recordset object

In MFC, an instance of class **CRecordset** or class **CDAORecordset**, or an instance of any class derived from these. See also record field exchange (RFX).

recursive

The ability of a function or a routine to call itself.

red, green, blue

A mixing model, or method of describing colors, in light-based media such as color monitors. RGB mixes percentages of the light-based primary colors (red, green, and blue) to create other colors. Windows defines these percentages as three 8-bit values called RGB values. Zero percentage of all three primaries, or an RGB value of (0,0,0), produces black and 100 percent of all three primaries, or an RGB value of (255,255,255), produces white. See also red, green, blue, alpha (RGBA).

red, green, blue, alpha

A model that augments the red, green, blue (RGB) method of combining colors in light-based media, with a fourth color component, alpha, which is used to control color blending. In Open GL, an alpha value of 1.0 corresponds to complete opacity and a value of 0.0 corresponds to complete transparency. See also red, green, blue (RGB).

redraw flag

A Boolean parameter to many Windows functions which, when cleared, prevents the content of the given window from being updated (redrawn). When set, this flag allows the window to be repainted with new information.

reentrant

Code written so that it can be shared by several programs (or processes within a single program) at the same time. When code is reentrant, one program or process can safely interrupt the execution of another program or process, execute its own code, and then return control to the first program or process in such a way that the first program or process does not fail or behave in an unexpected way.

reference

1 (noun) In C++, a form of pointer that can be used to access a variable by its address. When such a reference is declared, it must be associated with a variable. The reference becomes an alias for that variable. See also address-of (&) operator.

2 (verb) To access a variable, often said of elements in an array or in a record.

reference count

A count of the number of pointers that access, or make reference to, an object, allowing for multiple references to a single object. This number is decremented when a reference is removed; when the count reaches zero, the object's space in memory is freed.

referential integrity

In database management, a set of rules that preserves the defined relationships between tables when records are entered or deleted. Enforcing referential integrity would, for example, prevent a record from being added to a related table when there is no associated record in the primary table. See also schema.

region

- 1 In Windows-based applications, a rectangle, polygon, ellipse, or combination of two or more of these shapes used to define a part of the client window to be painted, inverted, filled with output, framed, or used for hit testing.
- 2 More generally, an area dedicated to, or reserved for, a particular purpose.

register

- 1 (noun) A small (typically 32-bit), named region of high-speed memory located within the microprocessor and used as a holding area for specific, sometimes critical, pieces of data related to activities going on within the system. For example, a register might be used to hold the results of an addition operation or to hold the address of a particular location in the computer's memory. See also register variable.
- 2 (verb) The process whereby an application or thread provides information about itself in a registration database or with a registration class. For example, an OLE server application registers with the OLE system registration database. See also registry.

register storage class

A declaration for function arguments and local variables requesting that the associated values be held in high-speed memory registers, if these registers are available. This storage class defaults to automatic when register storage is not possible. The Visual C++ compiler does not honor user requests for register variables; instead, it makes its own register choices when global optimizations are on. However, the compiler does honor all other semantics associated with the register keyword when this declaration is used. See also automatic storage class, static storage class.

register variable

In C/C++, a local variable that a program specifies to be stored in a high-speed memory register, if possible, using the keyword register. See also register storage class.

registration entry file

In OLE applications, a text file description of the classes supported by a server application. When a server application is installed in a system, the contents of its registration entry file are merged with the system registry. Registration entry files usually have a .REG filename extension.

registry

Or OLE system registry, Windows NT registry, system registration database. In 32-bit Windows, the database in which configuration information is registered. This database takes the place of most configuration and initialization files for Windows and new Windows-based applications. See also registration entry file.

registry key

A unique identifier assigned to each piece of information in the system registration database.

regular expression

A search string that uses special characters to match a text pattern in a file. For example, an asterisk (*) means zero or more occurrences of the preceding character; thus, ab*c matches ac, abc, abbc, abbbc, etc.

relational database

A type of database or database management system that stores information in tables—rows and columns of data—and conducts searches by using data in specified columns of one table to find additional data in another table.

relative link

Links that keep the hyperlink intact even if you move the file containing the hyperlink.

relative path

The location of a file, document, or other object as it relates to the current location. The relative path to a target object includes only that portion of the path that is different from the path of the source object. For example, the relative path from c:\animal\mammal\canine\fido to c:\animal\mammal\feline\fluffy is ..\..\feline\fluffy. See also fully qualified path.

relative time

Or time span. The difference between two absolute time values.

release version

A compiled version of a program that does not include the debugging and diagnostic features included in a build compiled in debug mode. For example, a release version will not include the source code contained in **ASSERT** macros. See also debug version.

relocation information

Information that tells the linker what instructions in the object file will need to be patched with addresses of other objects defined in other source files. Those addresses will be known when all the object files are linked together.

Remote Access Service

A Windows NT Server feature by which a single serial connection provides a remote workstation with host connectivity, NT file services, or Novell file and printing services (NWLink).

remote debugging

1 To debug an application written for, and running on, a platform other than the one on which you are running Visual C++ (the host machine). The remote machine is connected to the host machine through the serial port or, in the case of a Macintosh, an AppleTalk network connection. See also debug monitor.

2 More generally, to debug an application that is running on a computer other than the one on which the debugger resides.

remote machine

1 In the case of remote debugging, the machine on which the target application (the one being

debugged) is running, as opposed to the host machine, which is running the debugger. See also debug monitor.

2 Or remote computer. More generally, any computer that is not in the immediate vicinity, but which is accessible through some type of cable or communications link. See also local machine.

Remote Procedure Call

A widely used standard defined by the Open Software Foundation (OSF) for distributed computing. RPC enables one process to make calls to functions that are part of another process. The other process can be on the same computer or on a different computer on the network.

repeat count

A value specifying the number of times a keystroke is repeated as a result of the user holding down the key.

reserve size

The amount of a resource that is to be allocated (or committed) for a particular use. For example, in the header of a COFF file, the Windows NT–specific field Heap Commit Size specifies the size of the local heap that the linker and loader are to reserve for that file. Only the commit size is initially allocated; the rest is made available one page at a time, until the reserve size is reached.

resource

1 A program block, dialog box template, bitmap, font, or sound, for example, that can be used by more than one program or in more than one place in a program. The use of resources allows alteration of many features in a program without the necessity of recompiling the program from source code. See also resource type.

2 More generally, any part of a computer system or a network that can be allotted to a program or a process while it is running.

resource browser window

A window that displays the resources associated with a project. From the resource browser window, an application's resources can be added, deleted, or changed.

resource compiler

An application that creates a binary resource file based on the resource-definition (.RC) file. The resource compiler can also append binary resource data to an executable file and create a resource table in the executable file's header.

resource editor

A specialized utility program that can be used to add, edit and delete program resources.

resource fork

The portion of an Apple Macintosh file containing reusable items of information that the program can use during execution, such as blocks of code, icons, fonts, and digitized sounds. See also data fork.

resource identifier (ID)

In Microsoft Windows and in the Apple Macintosh Operating System, a number that identifies a particular resource within a given resource type—for example, a particular menu among many resources of type MENU that a program can use. The resource type, along with the resource ID,

uniquely defines every resource in an application. (Note that resources of different types can have the same resource ID.)

resource type

A category of structural or procedural resources in Microsoft Windows and in the Apple Macintosh Operating System, such as code, fonts, dialog boxes, or icons. The resource type, along with the resource ID, uniquely defines every resource in an application.

resource-definition file

Or resource script file. A text file containing descriptions of resources from which the resource compiler creates a binary resource file. For Microsoft Windows applications, resource-definition files usually have a .RC filename extension. For Apple Macintosh applications, such files are typically named with a .R extension and written with the Apple Rez script language. See also compiled resource file.

RESOURCE.H file

The default name for the header file that corresponds to an application's resource-definition file.

response file

A text file that can contain multiple options and filenames that would otherwise be typed on the command line or specified by using the CL environment variable. The options in the response file are interpreted as if they were present on the command line at the position of the response file invocation.

restored window

A window that has been returned to its preminimized or premaximized size and position.

result set

A collection of data returned by an SQL query on a database. A Known result set is when the database application knows the exact form of the SQL statements and, therefore, the form of the result set, at compile time. An Unknown result set is when the application does not know the exact form of the SQL statement at compile time and, therefore, cannot predict the format of these results prior to execution. See also recordset.

return code

Or result code, exit code. A predefined code used to report the outcome of a procedure or to influence subsequent events when a routine or process terminates (returns) and passes control of the system to another routine. The return code can indicate success or failure, or can even indicate a particular type of failure that is within the normal range of expectations. For example, the exception-handling functions in C++ generate return codes.

return type

The data type returned by a function. The type-specifier field of a function definition can specify any fundamental, class, structure, or union type, or **void**.

return value

The value that a function evaluates to in its calling expression.

Rez

A script language that is part of the Macintosh Programmer's Workshop and is used to specify the resources to be used by a Macintosh application. MRC.EXE, the Macintosh Resource Compiler, provides a Windows NT implementation of Rez.

RFX

When bulk row fetching is not implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also bulk record field exchange (Bulk RFX), dialog data exchange (DDX), and DAO record field exchange (DFX).

RGB

A mixing model, or method of describing colors, in light-based media such as color monitors. RGB mixes percentages of the light-based primary colors (red, green, and blue) to create other colors. Windows defines these percentages as three 8-bit values called RGB values. Zero percentage of all three primaries, or an RGB value of (0,0,0), produces black and 100 percent of all three primaries, or an RGB value of (255,255,255), produces white. See also red, green, blue, alpha (RGBA).

RGBA

A model that augments the red, green, blue (RGB) method of combining colors in light-based media, with a fourth color component, alpha, which is used to control color blending. In Open GL, an alpha value of 1.0 corresponds to complete opacity and a value of 0.0 corresponds to complete transparency. See also red, green, blue (RGB).

RGBA mode

In OpenGL, a color pixel data mode. An OpenGL context is in RGBA mode if its color buffers store red, green, blue, and alpha color components instead of color indexes.

rich-text format file

A file that contains encoded, formatted text and graphics for easy transfer between applications. The rich-text encoding format is commonly used by document-processing programs such as Microsoft Word for Windows and for generating online Help files. Rich-text format files usually have a .RTF filename extension.

right outer join

A database term that describes the relationship between two tables on which a query is run. If two tables are connected with a right outer join, the resulting recordset contains all records from the table on the right but only those records from the table on the left where the joined fields are equal. See also left outer join.

risk management

The total process of identifying, controlling, and eliminating or minimizing the effects of uncertain events, such as power failures or hacker activity, that may affect system resources. It includes risk analysis, cost benefit analysis, selection, implementation and test, security evaluation of safeguards, and overall security review.

RLE8 compression

An image compression format for device-independent bitmaps that encodes a sequence of identical pixels as a repeat count and a color value.

rollback

A database operation that allows recovery from changes made during a transaction that was canceled or failed. Rollback returns the records to the state they were in as of the last commit (saved transaction).

root

In a hierarchy of items, the one item from which all other items are descended. The root item has nothing above it in the hierarchy. See also inheritance hierarchy.

ROP

In computer graphics, a Boolean operation that combines either the values of the pixels in the selected pen and the destination bitmap (binary operation) or the values of the pixels in the source, the selected brush, and the destination (ternary operation). See also raster operation code.

rotation

A transformation that makes a displayed graphical object appear to have been turned (rotated) in space, relative to the coordinate-space origin.

router

A hardware device that connects two or more networks, or connects a network to the Internet. It converts addresses and sends on only those messages that need to pass to the other network.

rowset

In ODBC, one or more rows returned by a single fetch operation.

RPC

A widely used standard defined by the Open Software Foundation (OSF) for distributed computing. RPC enables one process to make calls to functions that are part of another process. The other process can be on the same computer or on a different computer on the network.

RTTI

In C++, a mechanism that allows the type of an object to be determined during program execution.

rubber-band selection

A selection method that allows a user to select multiple items by dragging a sizing rectangle around the items to be selected. Items within the region can be manipulated by the user. For instance, the user might drag or drop the selection into another container application.

run time

The point in time when a program is running, or the period of time needed to execute the program. See also compile time, link time.

run-time class information

Information about an object's class accessed at run time. This information is useful for doing extra type-checking or creating dynamic objects at run time. Classes derived from the MFC class **CObject** can have this functionality. See also run-time type information (RTTI).

run-time error

A math or logic error in a program that becomes apparent only at run time. Such an error may produce incorrect results or cause the program to behave unpredictably or crash.

run-time type information

In C++, a mechanism that allows the type of an object to be determined during program execution.

running state

In OLE, the status of a compound document object (either a linked or embedded item) when the object's application is running and it is possible to edit the object, access its interfaces, and receive notification of changes. See also loaded state.

Glossary S

sampling frequency

Or sampling rate. The rate at which samples of a physical variable, such as sound, are taken. The more samples taken per unit, the more closely the reconstructed result resembles the original.

scalable font

Any font that is defined by mathematical routines that can reproduce the outlines of each character at any size. Macintosh System 7, vector, PostScript, and TrueType fonts are all scalable; raster (bitmap) fonts are not.

scalar type

Or scalar data type. A data type that has a predictable and enumerable sequence of values that can be compared for greater-than/less-than relationships. In Visual C++, all arithmetic types, plus pointers, are considered scalar types. See also aggregate type.

scaled index base indirect operand

In Intel 80386-80486 assembly language, the second encoding byte of an extended memory operand.

scaling

A transformation that alters the apparent size of an object.

scan code

A device-dependent identifier that uniquely identifies each key on a keyboard. A keyboard generates two scan codes when the user types a key—one when the user presses the key and another when the user releases the key. See also virtual key code.

scan line

- 1 On a television or raster-scan computer monitor, one of the horizontal lines on the inner surface of the screen that is traced by the electron beam to form an image.
- 2 A row of pixels read by any scanning device, such as a full-page scanner or a fax machine.

schema

A description of the current structure of tables and views in a data source. The schema describes what columns are in each table, the data type of each column, and the relationships between tables. See also referential integrity.

schema number

In MFC serialization, the version of a class implementation, assigned to a class when the **IMPLEMENT_SERIAL** macro of the class is encountered. The schema number refers to the implementation of the class, not to the number of times a given object has been made persistent (usually referred to as the object version). Do not confuse this schema number with database terminology.

scope

In programming, the extent to which a given identifier (constant, variable, data type, routine) can be referenced within a program. See also class scope, file scope, function scope, function-prototype scope, local scope.

scope-resolution (::) operator

In C++, the operator with highest precedence, used to define the scope of the operand. The unary form of the operator, `::foo()`, is used to uncover or access a name that is at global scope and has been hidden by local or class scope. The binary form, `Bar::foo()`, is used to disambiguate names that are reused within classes, structures, or unions.

screen coordinates

A means of specifying the position of a point on the display screen in terms of vertical (y-coordinate) and horizontal (x-coordinate) displacement from the upper-left corner of the screen (origin). The position and size of a window can be described by one set of coordinates (x, y) that marks the point defining the upper-left corner of the window, and another set of coordinates (x', y') that marks the point defining the lower-right corner of the window.

script file

A type of program that consists of a set of instructions to an application or utility program. The instructions in a script are usually expressed using the application's or utility's rules and syntax, combined with simple control structures such as loops and **if-then** expressions. A resource-definition file and a batch (.BAT) file are two examples of script files.

scroll-bar code

A 2-byte value that indicates the user's scrolling request. For example, the constant SB_LEFT indicates a request to scroll to the far left and the constant SB_PAGELLEFT indicates a request to scroll one page to the left.

scroll-bar control

A control window that belongs to the SCROLLBAR window class. A scroll-bar control appears and functions like a standard scroll bar, but it is a separate window that receives direct input focus, indicated by a flashing caret displayed in the scroll box. Unlike a standard scroll bar, a scroll-bar control also has a built-in keyboard interface that enables the user to direct scrolling. See also standard scroll bar.

scrolling

The process of moving a document in a window to permit viewing of any desired portion.

scrolling range

The minimum and maximum values that a scroll bar can report.

SDI

A user interface architecture that allows a user to work with just one document at a time. Windows Notepad is an example of an SDI application. See also multiple document interface (MDI).

SDK

A set of libraries, header files, tools, books, on-line help and sample programs designed to help a developer create software.

sector

On a disk, the smallest contiguous physical unit for recording information. Multiple sectors make up a track.

Secure Sockets Layer

A protocol for providing data security layered between its service protocols (HTTP) and TCP/IP.

security attribute

A characteristic of a file or object that regulates its access and privileges by users or other objects. See also access token.

security descriptor

Contains the security information associated with an object. The information in security descriptors can include an owner, a primary group, a discretionary access-control list, and a system access-control list. This information is stored in the form of security identifiers (SIDs) and access-control lists (ACLs).

security identifier

A structure of variable length that uniquely identifies a user or group on all Windows NT implementations. See also access token, impersonation token, primary token, privilege.

security policy

The set of laws, rules, and practices that regulates how an organization manages, protects, and distributes sensitive information.

SEH

A mechanism for handling hardware- and software-generated exceptions that gives developers complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and computers. See also C++ exception handling.

selection statement

A statement that provides a means to conditionally execute sections of code. The **if** and **switch** statements are C/C++ selection statements.

semantics

The relationships between words or symbols and their intended meanings, or the rules governing these relationships. See also syntax.

semaphore

1 In Win32, a synchronization object that maintains a count between zero and a specified maximum value. A semaphore's state is signaled when its count is greater than zero and nonsignaled when its count is zero. The semaphore object is useful in controlling a shared resource that can support a limited number of users. It acts like a gate that counts the threads as they enter and exit a controlled area and that limits the number of threads sharing the resource to a specified maximum number.

2 More generally, a flag variable used to govern concurrent processes that share system resources.

separator

- 1 In Windows, a special type of menu item that appears as a horizontal line. A separator can be used in a pop-up menu to divide a menu into groups of related items.
- 2 Or separator code. A Unicode value used to indicate line breaks (0x2028) or paragraph breaks (0x2029).
- 3 More generally, a keyword, character, or white space used to separate components of a name, a number, or a group of fields. For example, the decimal point (.) in 123.456 and the backslash (\) in ROOT\SUBDIR are separators.

Serial Line Internet Protocol

A protocol for connecting to the Internet via a dial-up connection, such as with a modem.

serial port

An electrical connection to a computer through which data is transmitted in series, one bit after another.

serial transport layer

A data transport link established by connecting the serial ports of two machines. For example, a serial transport layer is established when an Apple Macintosh and a Win32 host are connected through their serial ports.

serialization

Or object persistence. In MFC, the process of writing or reading an object to or from a persistent storage medium, such as a disk file. The basic idea of serialization is that an object should be able to write its current state, usually indicated by the value of its member variables, to persistent storage. Later, the object can be re-created by reading, or deserializing, the object's state from storage.

serif

Any of the short lines or ornaments at the ends of the strokes that form a character in a typeface.

server

- 1 In a network, any device that can be shared by all users.
- 2 An application or a process that responds to a client request. See also client/server.

server application

An application that can create OLE items for use by container applications. Data in a server application can usually be copied, using the Clipboard or a drag-and-drop procedure, so that a container application can paste the data as an embedded or linked item. An application can be both a container and a server. See also container application, mini-server application.

server document

In OLE, a document created by a server application. See also compound document.

server item

An object that provides an interface between an OLE item and the server application and that is of a class derived from the MFC class **COleServerItem**. Server items, which are created and maintained by the server application, can be either linked or embedded. For linked items, the server item provides access to the data source, often in a different file. For embedded items, the server item handles the data stored in the container document and creates the server document. Server items also generate presentation data and handle the verbs (commands) associated with the OLE item.

service

In Win32, an executable object that is installed in a registry database maintained by the Service Control Manager. The executable file associated with a service can be started at boot time by a boot program or by the system, or it can be started on demand by the Service Control Manager. The two types of service are Win32 service and driver service.

session

- 1 In a Windows NT client/server network, a link between a workstation and a server. The session is established the first time a workstation makes a connection with a shared resource on the server and ends when all current connections between the workstation and the server are deleted.
- 2 In the ISO/OSI communications model, the protocol layer that provides a way for users to establish connections and transport data across those connections. See also transport layer.
- 3 More generally, the time during which two computers (or a computer and a terminal) maintain a connection, or the connection itself.

SGML

A set of rules and tags to mark the structure and content of a document, independent of the display medium.

shared library

In general, any code module that can be accessed and used by many programs. Shared libraries are used primarily for sharing common code between different executable files or for breaking an application into separate components, thus allowing easy upgrades. In the Visual C++ documentation, shared library usually refers to a code module that is an Apple Shared Library Manager (ASLM) file for the Apple Macintosh. In Windows, shared libraries are usually referred to as dynamic-link libraries (DLLs).

shared memory

Memory accessed by more than one process or thread in a multitasking environment. Processes or threads using this memory operate under a set of rules that prevent them from modifying the same addresses simultaneously.

shared resource

- 1 In a Windows NT client/server network, any local resource on the server that is made available to network users, such as directories, files, printers, or named pipes.
- 2 More generally, any device, data, or program that is used by more than one device, program, process, or thread.

sharing mode

A file opening mode that determines what, if any, read and write operations will be allowed when the file is being shared. See also shared resource, sharing violation.

sharing violation

An error that occurs when one process (or machine) attempts to access a file after a different process has requested that the server block access to the file. If an application opens the file in compatibility mode, a sharing violation results in a critical error. See also sharing mode.

shell

A piece of software, usually a separate program, that provides communication between the user and the operating system. For example, the Windows Program Manager is a shell program that interacts with MS-DOS.

SHIFT+F1 Help

Context-sensitive Windows Help that the user obtains by pressing the SHIFT and F1 keys together. SHIFT + F1 Help invokes a special Help mode in which the cursor turns into a Help cursor. The user can then select a visible object in the user interface, such as a menu item, toolbar button, or window. This opens Help on a topic that describes the selected item.

short integer

In 32-bit Visual C++, a 16-bit integer. Most compilers provide less storage space for a **short** than for an **int**; however, the ANSI standard guarantees only that a short integer is no longer than an integer, which is the machine's native (word) size.

shortcut

A fast way to perform an action such as selecting text or, more usually, opening a file, document, Web page, and so on. Usually represented by an icon on the desktop.

shortcut key

See accelerator key.

shortcut menu

A menu displayed within a window that provides quick access to frequently used commands that are also available from the main menu bar. The commands in a shortcut menu may change depending on the current state of the window.

show state

A collection of qualities that a windows has at a given time, including active or inactive; hidden or visible; and minimized, maximized, or restored.

SIB

In Intel 80386-80486 assembly language, the second encoding byte of an extended memory operand.

sibling

A node in a tree that is descended from the same immediate ancestor(s) as other processes or nodes. The node may represent any data structure, or a system object such as a window or a process.

sibling window

A child window that has the same parent window as one or more other child windows.

SID

A structure of variable length that uniquely identifies a user or group on all Windows NT implementations. See also access token, impersonation token, primary token, privilege.

side effect

A change of state, other than the obvious one, caused by a routine, function call, or assignment. Function calls can have side effects if they change the value of an externally visible item.

signature

- 1 In Win32, a 4-byte value that identifies an enhanced metafile.
- 2 More generally, a sequence of data used for identification, such as an identifier appended to an electronic mail message or in a fax.

signed integer

An integer data type that can be either positive or negative. The most significant bit is the sign bit, which is 1 for negative values and 0 for positive values. See also unsigned integer.

signed values

Values that can be negative or positive.

Simple Mail Transfer Protocol

A standard Internet protocol for sending e-mail documents.

Simple Network Management Protocol

Designed for requesting, packaging, and sending management information over a network. SNMP provides a common set of rules for programmers writing network management programs.

single document

A user interface architecture that allows a user to work with just one document at a time. Windows Notepad is an example of an SDI application. See also multiple document interface (MDI).

single-byte character set

A mapping of characters to their identifying numeric values, in which each value is 1 byte wide. The ANSI and OEM character sets are single-byte character sets. See also multibyte character set (MBCS), Unicode.

single-line edit control

An element of the Windows user interface that allows the user to enter and edit a single line of text. See also multiline edit control.

size box

A small rectangle the user can manipulate to change the size of a window.

sizing border

A type of window border that enables the user to size the window by clicking and dragging the border.

sizing handles

A mechanism for changing the size of a bitmap or a control. Active sizing handles are solid squares; if a sizing handle is a hollow square, the object cannot be resized along that axis.

skeleton application

Or starter application. A default application created by AppWizard that runs, opens and closes windows, and allows other operations on the windows. You add the necessary code to implement the functionality needed for your own application.

slider control

A window containing a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the control sends notification messages to indicate the change.

SLIP

A protocol for connecting to the Internet via a dial-up connection, such as with a modem.

small memory model

A memory model with only one code segment and only one data segment.

smart pointer

In C++, an object that implements the functionality of a pointer and additionally performs some action whenever an object is accessed through it. Smart pointers are implemented by overloading the pointer-dereference (->) operator.

SMTP

A standard Internet protocol for sending e-mail documents.

snapshot

- 1 In MFC, a recordset that reflects a static view of the data as it existed at the time the snapshot was created. See also dynaset, recordset.
- 2 Or screen dump. A copy of all or part of the display screen as it appears at a given instant.
- 3 More generally, a copy of an object's state or appearance at a given time.

SNMP

Designed for requesting, packaging, and sending management information over a network. SNMP provides a common set of rules for programmers writing network management programs.

socket

An object that represents an endpoint for communication between processes across a network transport (TCP/IP or AppleTalk, for example). Sockets have a type (datagram or stream) and can be bound to a specific network address. Windows Sockets provides an API for handling all types of socket connections in Windows. See also datagram socket, stream socket, transport protocol.

software development kit

A set of libraries, header files, tools, books, on-line help and sample programs designed to help a

developer create software.

solid brush

A logical brush that contains 64 pixels of the same color. See also null brush.

sort order

The order in which a set of records or other data objects are to be sorted, or the function that defines this order. Possible sort orders for an array of strings, for example, could include lexicographic order or ascending order by length.

source character set

The set of legal characters that can appear in source files. For Microsoft C and C++, the source set is the standard ASCII character set. The source character set and execution character set include the ASCII characters used as escape sequences. See also execution character set.

source code

Human-readable statements written in a high-level programming language, or assembly language. See also object code.

source code editor

A text editor that may provide special formatting features which make it easier to generate readable, syntactically correct source code. For example, a source code editor may automatically indent blocks of code, check for balanced parentheses and brackets, or highlight keywords.

spawn

- 1 (noun) A family of C run-time library functions that create and execute new child processes.
- 2 (verb) To create a child process (or thread, in cases where multiple threads are allowed).

spline

In computer graphics, a curve calculated by a mathematical function that connects separate points with a high degree of smoothness.

SQL

A database sublanguage used to query, update, and manage relational databases.

SSL

A protocol for providing data security layered between its service protocols (HTTP) and TCP/IP.

stack

- 1 A data structure implemented as a LIFO (last in, first out) list so that the last item added to the structure is the first item removed.
- 2 A region of reserved memory, organized as a stack, in which programs temporarily store status data such as procedure and function call return addresses, passed parameters, and local variables. See also heap.

stack allocation

The amount, in bytes, of space reserved for a program's status data such as procedure and function call return addresses, passed parameters, and local variables. See also stack frame.

stack frame

Or frame allocation. An area of memory set up whenever a function is called that temporarily holds the arguments to the function as well as any variables that are defined local to the function. There are two key characteristics of frame allocations. First, when a local variable is defined, enough space is allocated on the stack frame to hold the entire variable, even if it is a large array or data structure. Second, frame variables are automatically deleted when they go out of scope.

stack overflow

An error condition caused by attempting to push an item onto a stack that is full, meaning that all of the memory allocated for that stack has been used. See also stack underflow.

stack probe

A short routine, called on entry to a function, to verify that there is enough room in the program stack to allocate local variables required by the function.

stack size

The amount of memory, in bytes, allocated to a stack.

stack underflow

An error condition caused by attempting to pop an item from an empty stack. See also stack overflow.

stand-alone code

For the Apple Macintosh and Power Mac, code that implements resources as a shared library, with some of the functionality of dynamic-link libraries. See also Apple Shared Library Manager (ASLM).

standard control

One of the controls provided by Microsoft Windows. These controls include buttons of several kinds, static- and editable-text controls, scroll bars, list boxes, and combo boxes. See also custom control.

standard conversion

In C++, the conversion of objects of one fundamental type to another type. For example, converting an object of integral type to a shorter signed or unsigned integral type. Standard conversion can result in loss of data if the value of the original object is outside the range that can be represented by the shorter type.

standard error device

The device to which a program sends its error messages unless the error output is redirected. Normally, the standard error device is the console.

Standard Generalized Mark-up Language

A set of rules and tags to mark the structure and content of a document, independent of the display medium.

standard input device

The device from which a program reads its input unless the input is redirected. In normal operation, the standard input device is the keyboard.

standard input/output (I/O)

In C, the input and output functions declared in the STDIO.H header file. See also standard input device, standard output device, standard error device.

standard output device

The device to which a program sends its output unless the output is redirected. In normal operation, the standard output device is the console.

standard resource

A resource whose format is defined and recognized by Windows. Standard resources include icons, cursors, menus, dialog boxes, bitmaps, fonts, keyboard accelerator tables, message-table entries, string-table entries, and version data. See also custom resource.

standard scroll bar

One of two ways to include a scroll bar in a window. A standard scroll bar is located in the nonclient area of a window. It is created with the window and displayed when the window is displayed. The sole purpose of a standard scroll bar is to enable the user to generate scrolling requests for viewing the entire content of the client area.

start page

A page the user chooses as the opening page of the Internet or a Web site.

starter files

In Visual C++, a set of files created by AppWizard that, when compiled, implement the basic features of a Windows application. The starter files consist of C++ source files, resource files, header files, and a project file. See also skeleton application.

startup code

The portion of the program code that gets an application up and running. Startup code interprets command-line arguments, creates and initializes global variables, opens standard streams, and so forth.

state flags

Values, often return values from functions, that specify the condition of an interface component such as a checkbox.

static control

A control that enables an application to provide the user with certain types of text and graphics that require no response. Applications often use static controls to label other controls or to separate a group of controls.

static cursor

An ODBC cursor that appears to be fixed (static) from the perspective of the data in the underlying tables. Changes made by other users are not detected by the static cursor until it is closed and reopened. See also [snapshot](#).

static data

Data declared with the keyword **static**. Static data can include initialized variables defined outside of functions, static variables within functions, explicit strings, and [floating-point](#) numbers. Static data also sometimes refers to the area in memory where static data resides. See also [statically allocated buffer](#).

static data member

Or [static member variable](#). A data [member](#) that is declared within the [scope](#) of a class but which is actually a separate [object](#). The [definition](#) of the static data [member](#) is performed elsewhere in the program, and only one copy of the [member](#) exists, no matter how many objects of the class exist.

static extent

A property of [global](#) data objects (both **static** and **extern**), [local](#) static objects, and [static data members](#) of C++ classes that have the following characteristics:

- Only one copy of the data is maintained for all objects.
- The objects retain their location in memory from the time they are created until they are destroyed.

static link

A program [link](#), to a library or to an [object](#), that is established at [link](#) time. When an application uses a function from a static-link library, the linker copies the code for that function into the application's executable file. See also [dynamic link](#).

[static splitter](#) window

A split-window style in which the [panes](#) are created when the window is created, and the order and number of [panes](#) never change. The [panes](#) are separated by a splitter bar that the user can drag to change the relative sizes of the [panes](#). See also [dynamic splitter](#) window.

static storage class

In C++, the storage class for objects and variables that exist and retain their values throughout the execution of the entire program. All [global](#) objects have static storage class. Local objects and class members can be given static storage class by explicit use of the **static** storage class specifier. See also [automatic storage](#) class.

static-link library

A library file that is linked into the program when the executable file is built. Static-link library files usually have a .LIB filename extension. See also [dynamic-link library](#) file, library file.

statically allocated buffer

Or [static buffer](#). A portion of memory that is allocated when a module is loaded and is deallocated when the module leaves memory.

status bar

A control bar at the bottom of a window, with a row of text output panes. The status bar is usually used as a message line (for example, the standard menu help message line) or as a status indicator (for example, the CAP, NUM and SCRL indicators). See also dialog bar.

status code

A value used to report on the current status of an object, event, or process, or to reflect the outcome of an operation.

storage class

Or storage duration. In C/C++, determines whether a variable (or object, in C++) has a static (or global) lifetime, in which case it is stored in the same memory location throughout the execution of the program, or an automatic (or local) lifetime, (in which case it is allocated new storage each time execution control passes to the block in which it is defined. See also automatic storage class, register storage class, scope, static storage class.

storage object

An object type used in OLE to implement compound files. Storage objects are analogous to directories in that they can contain other storage objects, or they can contain stream objects, which are analogous to files. See also compound file.

stream file object

A virtual file representing on-disk data associated with a file, some of which may not be part of the physical file that backs a file object. For example, a stream file object makes it possible to cache the extended attributes (EAs) or access-control list (ACL) for a file object together with the file's data.

stream I/O

Or iostream. In C++, the input and output functions, declared in IOSTREAM.H, that transfer data from and to files and devices. Stream I/O functions treat data as a stream of individual characters and provide buffering. The predefined object cout represents the standard output stream, cin represents the standard input stream, and cerr represents the standard error stream.

stream object

One of the object types used in OLE to implement compound files. Stream objects store data of any type. See also compound file, storage object.

stream socket

A connection-oriented socket that provides a bidirectional, sequenced, and unduplicated flow of data without record boundaries. Receipt of stream messages is guaranteed, and streams are well-suited to handling large amounts of data. Stream sockets are appropriate, for example, for implementations such as file transfer protocol (FTP), which facilitates transferring ASCII or binary files of arbitrary size. See also datagram socket, transport protocol.

streaming

The process of transferring information from a storage device, such as a hard disk or CD-ROM, to a device driver. Rather than transferring all the information in a single data copy, the information is transferred in smaller parts over a period of time, typically while the application is performing other

tasks.

strikeout

A font effect that adds a horizontal line through one or more characters.

string

A data structure composed of a sequence of characters identified with a symbolic name. In C/C++, a string is terminated with a null character ('\0').

string constant

In C/C++/Java programming, a list of characters enclosed in double quotes in source code.

string identifier (ID)

A 16-bit identifier that Windows uses to locate a string in a string-table resource. The upper 12 bits of the identifier specify the block in which the string appears. The lower 4 bits specify the ordinal location of the string within the block. See also string table.

string literal

Or **literal string**, **string constant**. A string of characters enclosed with double quotation marks ("). Any character from the source character set is allowed, except that a double quotation mark inside the string must be preceded by the backslash, or escape, character (\). Like other constants, string literals do not change in a program. See also character constant.

string name

An identifier for a class, version, or resource, in the form of a human-readable character string.

string resources

In 32-bit Windows, null-terminated Unicode strings that are stored in the resource file string table. Each string is made up of a series of strings whose ordinal position is used as the string ID.

string table

1 A Windows resource that contains a list of identifiers, values, and captions for the strings used in an application's **framework**. For example, the status bar prompts are located in the string table. When the resource **compiler** converts a string table specified in a resource-definition file, it separates it into blocks of 16 strings and stores them as individual resources. See also string resources.

2 More generally, any data structure used to store character strings. Typically, a string table is implemented as a hash table.

structure

1 In C, an aggregate data type that can contain constants, variables, and other structures. In C++, a structure can also contain functions and all of a structure's members are implicitly **public**.

2 More generally, a collection of data elements.

structured exception

A mechanism for handling hardware- and software-generated exceptions that gives developers complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and computers. See also C++ exception handling.

Structured Query Language

A database sublanguage used to query, update, and manage relational databases.

structured storage

An OLE model that allows objects to control their own data storage, loading directly from and saving directly to disk.

stub

An interface-specific object that unpackages the parameters for that interface after they are marshaled across the process boundary, and makes the requested method call. The stub runs in the address space of the receiver and communicates with a corresponding proxy in the sender's address space.

stub file

Or stub program. In Windows, an MS-DOS executable file added to the beginning of a segmented executable file and invoked if a user tries to run a Windows program from the MS-DOS prompt. The stub may display some error message such as This program requires Microsoft Windows.

style

A value, or set of values, that defines the outward appearance and behavior of an object, such as a window, control, or document. See also window style.

style bit

An individual bit of the 16-bit style parameter that pertains to a single style attribute. For example, the WS_VISIBLE style bit, when set, determines whether a particular window is visible to the user.

subaddress

The part of a URL that goes to a specific place in a file, such as a bookmark, slide, and so on.

subclass

Or derived class. The class that is derived from another class. A subclass inherits state and behavior from its superclass or superclasses in the form of variables and methods.

subclassing

In Windows programming, a technique that allows an application to intercept and process messages sent or posted to a particular window before the window has a chance to process them. By subclassing a window, an application can augment, modify, or monitor the behavior of the window.

subexpression

An expression that is part of a larger expression. For example, (a+b) is a subexpression of (a+b)*c.

sublanguage

Or secondary language. In the localization of Windows-based programs, a variant of the primary language, defined by the locale. For example, if English is the primary language, American, British, Australian, New Zealand, Canadian, and Ireland are the possible locales that determine the sublanguage. See also language identifier (ID).

subobject

- 1 The portion of a Windows object that is completely described by a base class.
- 2 Generally, an object within an object. For example, a cell could be considered a subobject of a spreadsheet object.

subscript

- 1 In programming, a value enclosed in brackets ([]) that indicates either the number of elements in an array in the array declaration or the offset position within an array.
- 2 In printing, one or more characters printed slightly below the bottom edge of the surrounding text.

subscript ([]) operator

An operator that indicates that the name preceding the operator is an array, or that designates a subscript into the array. For classes that have overloaded the operator, the behavior of this operator is class-specific.

substring

A string that is part of a longer string. For example, the string cat is a substring of the string catamaran.

subsystem

A system other than, and usually subordinate to, the primary system. A subsystem can have its own memory allocation and internal functions. OLE 2.0 and DEBUG are examples of subsystems, and POSIX is a subsystem of Windows NT that can run UNIX applications.

superclass

Or base class. A class which is the base (or parent) class for another class; a class from which another is derived, either directly or indirectly. The superclass provides state and behavior to the subclass which the subclass may modify by overriding selected methods.

suspend count

A record of the number of active operations that require a thread to momentarily suspend execution of user-mode code. Starting one of these operations increments the suspend count; ending the operation decrements the count. When the suspend count equals zero, the thread resumes execution.

swap file

Or paging file. In Windows, the disk file that holds the active system and application memory pages that are not currently present in main memory (RAM). See also virtual memory.

symbol

- 1 A character other than the standard alphanumeric characters. It usually refers to algebraic, scientific, or linguistic characters not found on the keyboard.
- 2 In programming, a name that represents a register, an absolute value, or a memory address (relative or absolute).
- 3 To a compiler, a variable, function name, or other identifier.
- 4 In Visual C++, a resource identifier that consists of a text string (name) mapped to an integer value. A symbol provides a way to refer to resources and user-interface objects, both in source

code and in the resource editors.

symbolic-debugging information

A map of the source code and all the identifiers (variables, function names, and so on) created at compile time for use by the debugger. See also program database (.PDB) file.

synchronization object

An object whose handle can be specified in one of the wait functions to coordinate the execution of multiple threads. The state of a synchronization object is either signaled, which can allow the wait function to return, or nonsignaled, which can prevent the function from returning. More than one process can have a handle of the same synchronization object, making interprocess synchronization possible. See also mutex object, semaphore.

synchronous operation

- 1 In Windows programming, a task that requires the thread that initiated the operation to suspend activity until the task is completed. See also asynchronous operation, atomic operation.
- 2 In hardware, an operation that proceeds under control of a clock or timing mechanism.

synchronous processing

In ODBC, a method of processing transactions in which the database driver does not return control to an application until a function call completes. See also asynchronous processing.

syntax

The grammar of a particular language, the rules governing the structure and content of the statements. See also semantics.

System 7

An operating system for Apple Macintosh computers.

system database

A file, read at startup, that contains information about the users in a workgroup, such as account names, user preference information, and passwords.

system font

The font used by the operating system to display messages. The system font is the default font for resources.

system modal dialog box

Or system modal message box, system modal window. A dialog box that prevents the user from doing anything else in Windows until the dialog box is cleared, usually by choosing a pushbutton marked either OK or Cancel. Use a system modal dialog box to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory).

system palette

A representation of the device's physical palette. The system palette contains the RGB values for all colors that can currently be displayed or drawn by the device.

system time

The current time on the system's real-time clock. The system time structure contains values for the year, month, day, hour, minute, second, and millisecond.

SYSTEM.INI file

A Windows initialization file that contains the settings needed to configure Windows to a system's particular components.

Glossary T

tab order

The order in which the TAB key moves the input focus from one control to the next within a dialog box. Usually, the tab order proceeds from left to right in a dialog box, and from top to bottom in a radio group.

tab stop

One of the points in a line of text or a control in a group of controls (in a dialog box, for example) that the user can move to by pressing the TAB key. See also tab order.

tab-delimited report

A data file in which the elements are separated by tab characters.

tag

1 In C/C++, an optional identifier are part of structure, union, and enumeration type specifiers and, if present, always immediately follow the reserved words **struct**, union, or **enum**. The tag names must be distinct from all other structure, enumeration, or union tags with the same visibility.

2 Text in angle brackets that represents HTML markup. Web browsers display text and graphic elements based on the tags an author uses. The tag itself is not displayed by the browser.

tail

The last element in a linked list.

TAPI

A set of functions that is part of the Win32 API that lets a computer communicate directly with telephone systems.

target

The objective, or destination, of a computer command or operation. For example, the target machine in a remote debugging operation is the machine running the application that is being debugged.

task handle

One of two handles that Windows creates for each task running in the system. The task handle is the handle to the task database (TDB), which contains information about the task's queue, module handle, and so forth. See also instance handle.

TCP/IP

A set of transport protocols for the Internet that provides both connection-oriented (TCP) and connectionless (IP) data transfer. Commonly made up of four protocols: IP, TCP, UDP, and ICMP. See also transport protocol, User Datagram Protocol (UDP).

Telephony Application Programming Interface

A set of functions that is part of the Win32 API that lets a computer communicate directly with telephone systems.

teletype network

Terminal-emulation protocol for remote login over the Internet. Also refers to a UNIX program that uses the protocol (often written TELNET in that case).

telnet

Terminal-emulation protocol for remote login over the Internet. Also refers to a UNIX program that uses the protocol (often written TELNET in that case).

template

1 In C++, a keyword that allows polymorphism with respect to different types, by passing the data type as a parameter to the code body.

2 More generally, a form or blueprint for an object that contains information about the default properties of that object. For example, a Microsoft Word document template may contain text, formatting, and graphics information as well as macros and AutoText entries.

template class

A C++ class that is instantiated by providing a specific data (or class) type to a template. The compiler builds a class to process data of that type according to the specifications of the template. The Microsoft Foundation Class Library uses template classes to implement the standard collection classes.

temporary object

An object that is created when needed and destroyed after the reference object to which it is bound is destroyed.

temporary window

A window that an application creates for some temporary purpose. For example, a dialog box is a temporary window created to receive user input.

termination

The ending of a thread, process, or program.

termination functions

Functions called internally by MFC member functions when there is a fatal error, such as an uncaught exception that cannot be handled. In MFC, **AfxAbort** is the default termination function. The C/C++ Run-Time Library provides the _abort() function for non-MFC code. Most dynamic-link libraries register termination functions as well.

termination handler

A mechanism by which a developer ensures that a block of termination code is executed, so that resources such as memory, handles, and files are properly closed regardless of how a section of code finishes executing. A termination handler consists of a guarded body of code and a termination block. See also C++ exception handling, structured exception handling (SEH), **try** block.

ternary operator

An operator that takes three operands—for example, the conditional-expression (**? :**) operator in

C/C++. See also binary operator, unary operator.

text editor

A program used to manage, edit, and print text files.

text file

A human-readable file composed of text characters. A text file is usually identified by a file extension of .TXT. See also binary file, rich-text format (.RTF) file.

text mode

One of two modes for file I/O operations specified in the file-opening function. In text mode, control characters that specify the end of a line are normalized during I/O operations. In binary mode, no translation occurs.

text-alignment flag

An indicator that determines how text output functions position a string of text on a display or device.

thread

The basic entity to which the operating system allocates CPU time. A thread can execute any part of the application's code, including a part currently being executed by another thread. All threads of a process share the virtual address space, global variables, and operating-system resources of the process.

thread local storage

A Win32 mechanism that allows multiple threads of a process to store data that is unique for each thread. For example, a spreadsheet application can create a new instance of the same thread each time the user opens a new spreadsheet. A dynamic-link library that provides the functions for various spreadsheet operations can use thread local storage to save information about the current state of each spreadsheet (row, column, and so on).

thread switch

A change of context from one thread to another, either inside a single process or across processes.

three-state check box

A square box button control that can have one of three states, usually checked, unchecked (cleared), or indeterminate (grayed).

throw expression

In C++, a statement that transfers program control to a catch block in order to handle an exception. See also C++ exception handling, catch block, try block.

thumbnail representation

1 Or thumbnail view. In OLE, the reduced image of a document stored within an OLE compound file.

2 In general, a greatly reduced version of an image that contains just enough detail for the image to

be recognizable. Thumbnails are often used in a gallery view to allow the user to browse and select from a collection of images.

thunk

A small section of code that performs a translation or conversion during a call or indirection. For example, a thunk is used to change the size or type of function parameters when calling between 16- and 32-bit code.

time-out value

Or time-out delay. The maximum amount of time one entity will wait for another entity to complete a transaction. For example, in ODBC a query time-out value determines the amount of time the database engine will wait for a query's action to complete.

timer identifier (ID)

A value that identifies a timer or the events associated with a timer.

timestamp

A value that specifies the time data was created, modified, accessed, or received. In files, the timestamp may also specify when the data was committed to disk.

TLS

A Win32 mechanism that allows multiple threads of a process to store data that is unique for each thread. For example, a spreadsheet application can create a new instance of the same thread each time the user opens a new spreadsheet. A dynamic-link library that provides the functions for various spreadsheet operations can use thread local storage to save information about the current state of each spreadsheet (row, column, and so on).

token

- 1 In a source program, the basic element recognized by a compiler. Keywords, identifiers, constants, string literals, and operators are examples of tokens.
- 2 A group of security attributes created when a user logs on to the operating system. See also access token, primary token, impersonation token, privilege, security identifier (SID).

tool tip

A tiny pop-up window that presents a short description of a toolbar button's action. Tool tips are displayed when the user positions the mouse over a button for a period of time.

toolbar

A control bar based on a bitmap that contains a row of button images. These buttons can act like pushbuttons, check boxes, or radio buttons. See also dialog bar, status bar

top-level window

A window that has no parent window, or whose parent is the desktop window.

topmost window

The window that overlaps all the other windows even if it is not the active or foreground window.

trace message

Or trace output. An error or diagnostic message employed in debugging to provide information about where in the program execution a problem occurred. In some cases, trace output can provide advance warning about problems that are about to occur.

tracker

In OLE, a border, or adornment, for OLE items that provides a visual cue about the current status of the item. By using different tracker styles, OLE items can be displayed with hatched borders, resize handles, or a variety of other visual effects.

tracking

- 1 In user-interface control, to cause an on-screen displayed symbol, such as a pointer, to match the movements of a mouse or other pointing device.
- 2 In data management, to follow the flow of information through a manual or an automated system (a tracking tool).
- 3 In data storage and retrieval, to follow and read from a recording channel on a disk or a magnetic tape.
- 4 In general, the act of following a path.

tracking size

The window size (maximum or minimum) that the user can produce by dragging a sizing border or splitter bar.

trail byte

In a double-byte or multibyte character set, the second byte of a two-byte character. See also lead byte.

transacted mode

A file-access mode that buffers all changes to a document and writes the changes to disk or discards them only when an explicit commit or revert request occurs. In this way, the original file can be reverted to. See also direct mode, rollback, transaction.

transaction

In data management, a means of completing an all or nothing series of changes to a file. If one change fails, or if there is a system failure during the transaction, the file reverts back to its original state before the transaction began. See also rollback.

transient

In Java, a variable type qualifier denoting that the indicated variable is not part of the persistent state of the object.

translation

- 1 In programming, to convert a program from one language to another—for example, to convert C source code to object code.
- 2 In graphics, to move an image horizontally, vertically, or both, without rotating the image.
- 3 More generally, to convert from one form to another—for example, to translate a scan code into a key code.

translation phase

One of the steps a compiler follows in creating an executable program. In C/C++, these steps include character mapping, line splicing, tokenizing, preprocessing, character-set mapping, string concatenating, translating, and linking.

translation unit

- 1 A single source file, together with all of its include files, supplied as input to the compiler, which combines and translates the files to produce an object file. See also compilation unit.
- 2 In C/C++, a sequence of tokens that the compiler generates during the preprocessor phase. The translation unit incorporates code from the preprocessor directives, such as the **#include** and **#define** directives, into the source code from a single file and excludes any code removed by conditional compilation directives.

Transport Control Protocol/Internet Protocol

A set of transport protocols for the Internet that provides both connection-oriented (TCP) and connectionless (IP) data transfer. Commonly made up of four protocols: IP, TCP, UDP, and ICMP. See also transport protocol, User Datagram Protocol (UDP).

transport layer

- 1 In remote debugging, a data link established between the host machine and the target machine. See also serial transport layer.
- 2 The layer in the ISO/OSI communications model that is responsible for quality of service and accurate delivery of information. Among other services, the transport layer handles error detection and correction.

transport protocol

A set of conventions that govern how data is transported across networks. In a connection-oriented transport protocol, such as Transmission Control Protocol (TCP), applications are required to establish a virtual circuit before data transfer can take place. In a connectionless transport protocol, such as User Datagram Protocol (UDP), an established circuit is not required for data transfer and an application need only open and bind a socket in order to send and receive data.

trigraph

In C/C++, a sequence of two question marks followed by a punctuation character, which the compiler replaces with another character. For example, the compiler will replace the trigraph ??- with the character ~. Trigraphs allow C programs to be written using only the ISO Invariant Code Set, which is a subset of the 7-bit ASCII character set.

TrueType font

A scalable outline font whose glyphs are stored as a collection of line and curve commands plus a collection of hints. Windows uses the line and curve commands to define the outline of the glyph and uses the hints to adjust the length of the lines and the shapes of the curves to correct irregularities in their shapes that occur during rasterization. See also raster font, vector font.

try block

A guarded body of code in a **try-except** frame-based exception handler or **try-finally termination handler**. See also catch block, throw expression.

type cast

An explicit conversion of a variable, structure, object, or expression from one data type to another.

type checking

The examination by a compiler or interpreter of the operations in a program to make sure that the correct data types are being used. See also run-time type information (RTTI).

type declaration

A declaration in a program that specifies the characteristics of a new data type, usually by combining more primitive existing data types. See also data declaration.

type definition

1 A declaration that introduces a name which, within its scope, becomes a synonym for a type or a derived type. A type definition is usually used to construct shorter or more meaningful names for types already declared or to encapsulate implementation details that may change.

2 A definition that describes the characteristics of an object—for example, a Windows type definition describes the dimensions, colors, behavior, and position for an object of a particular window class.

type library file

Or OLE library. An OLE compound document file containing standard descriptions of data types, modules, and interfaces that can be used to fully expose objects for OLE Automation. The type library file usually has a .TLB filename extension and can be used by other applications to get information about the automation server.

type modifier

A keyword that modifies the data type that follows—for example, **unsigned** can be used to modify an integral data type such as **int**.

type qualifier

A keyword that provides specific properties to an identifier. The **const** type qualifier declares an object to be nonmodifiable. The **volatile** type qualifier declares an item whose value can legitimately be changed by something beyond the control of the program in which it appears, such as a concurrently executing thread.

type safety

The assurance that a given function will be not presented, at run time, with data of a type it cannot handle. Type safety is assured through type checking and/or by the use of template classes that are designed to operate on data of many types. See also run-time type information (RTTI).

type-safe collection

A collection class that enforces type safety on data or objects. For example, a type-safe collection can be implemented by using one of the MFC template-based classes such as **CArray** or **CList**, which can store data of any type.

typed pointer

A pointer to a specified type.

typeface name

The name of a font—for example, Times New Roman.

Glossary U

UDP

A connectionless transport protocol that forms a user interface to the Internet Protocol (IP).

UDT

In OLE, a set of interfaces that allow data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data.

unary operator

An operator that takes only one operand—for example, the increment (++) operator. See also binary operator, ternary operator.

unbalanced parentheses

The number of opening parentheses does not equal the number of closing parentheses.

UNC

The standard format for paths that include a local area network file server, as in `\server\share\path\filename`.

undecorated name

In C++, the form that an identifier has in the source code (as a string of human-readable characters) as opposed to its decorated name, which consists of symbols that are meaningful only to the compiler and the linker.

underflow

Or loss of precision. A condition in which a mathematical calculation produces a result too near to zero to be represented by the range of binary digits available to the computer for holding that value in the specified precision. See also loss of significance.

underhang

The amount of white space between the edge of a character cell and the black part of the glyph. See also ABC width, overhang.

undo flag

A value, maintained by the edit control, that indicates whether an application can reverse the most recent operation on the edit control (to undo a text deletion, for example).

Unicode

A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard. Windows NT uses Unicode exclusively at the system level.

Uniform Data Transfer

In OLE, a set of interfaces that allow data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data.

Uniform Resource Identifier

A variant of Uniform Resource Locator (URL).

Uniform Resource Locator

The address of a resource on the Internet. URL syntax is in the form `protocol://host/localinfo`, where `protocol` specifies the means of fetching the object (such as HTTP or FTP), `host` specifies the remote location where the object resides, and `localinfo` is a string (often a file name) passed to the protocol handler at the remote location. Also called Universal Resource Locator, Uniform Resource Identifier (URI).

union

- 1 In C/C++, a user-defined data type that can hold values of different types at different times. It is similar to a structure except that all of its members start at the same location in memory. A union variable can contain only one of its members at a time. The size of the union is at least the size of the largest member.
- 2 The keyword used to define union variables.
- 3 Two or more objects joined into one.

Universal Naming Convention

The standard format for paths that include a local area network file server, as in `\server\share\path\filename`.

universally unique identifier

Or globally unique identifier (GUID). A 128-bit value that uniquely identifies objects such as OLE servers, interfaces, manager entry-point vectors, and client objects. Universally unique identifiers are used in cross-process communication, such as RPC, and OLE.

unresolved external

An error report that results from the linker not being able to identify and link to a data type or function that is used in source code. Unresolved externals usually mean that a DLL or object file was missing during linking.

unsigned integer

A data type that can only hold a whole number with a value greater than, or equal to, zero. In this implementation, the maximum value that an unsigned integer can hold is 0xFFFFFFFF (4,294,967,295).

update handler function

A function that administers changes made to an object or data file to make it more current.

update option

A choice that is made about when and how a system, object, or data file will be changed to make it more current. For example, OLE linked objects, or items, can be automatically updated whenever possible, when the source document is saved, or only on request from the client application.

update region

Or invalid region. In Windows, identifies the portion of a window that is out-of-date or invalid and

in need of repainting.

updates

Actions or processes that make a system, object, or data file more current.

upper bound

The upper limit in an allowable range of values.

URI

A variant of Uniform Resource Locator (URL).

URL

The address of a resource on the Internet. URL syntax is in the form `protocol://host/localinfo`, where `protocol` specifies the means of fetching the object (such as HTTP or FTP), `host` specifies the remote location where the object resides, and `localinfo` is a string (often a file name) passed to the protocol handler at the remote location. Also called Universal Resource Locator, Uniform Resource Identifier (URI).

Usenet

Another name for Internet Newsgroups. A distributed bulletin board system running on news servers, UNIX hosts, online services and bulletin board systems. Collectively, all the users who post and read articles to newsgroups. The Usenet is international in scope and is the largest decentralized information utility, including government agencies, universities, high schools, and organizations of all sizes as well as millions of stand-alone PCs.

User Datagram Protocol

A connectionless transport protocol that forms a user interface to the Internet Protocol (IP).

User Network

Another name for Internet Newsgroups. A distributed bulletin board system running on news servers, UNIX hosts, online services and bulletin board systems. Collectively, all the users who post and read articles to newsgroups. The Usenet is international in scope and is the largest decentralized information utility, including government agencies, universities, high schools, and organizations of all sizes as well as millions of stand-alone PCs.

user-defined message

Any message that is not a standard Windows message.

user-interface object

In Windows, an object that provides functionality to the user-interface. For example, menu items, toolbar buttons, and accelerator keys are all user-interface objects.

user-interface thread

In Windows, a thread that handles user input and responds to user events independently of threads executing other portions of the application. User-interface threads have a message pump and process messages received from the system. See also worker thread.

USRDLL

A version of the Microsoft Foundation Class Library used to statically link MFC into a stand-alone user DLL so that the user DLL can incorporate some of the MFC classes.

UTC

A global time standard equivalent to Greenwich mean time (GMT).

UUID

Or globally unique identifier (GUID). A 128-bit value that uniquely identifies objects such as OLE servers, interfaces, manager entry-point vectors, and client objects. Universally unique identifiers are used in cross-process communication, such as RPC, and OLE.

Glossary V

validation rules

A set of functions that dialog data verification (DDV) employs to validate user entries in a dialog control. For example, the function `DDV_MaxChars` could validate that the string entered in the text-box control is not greater than a specified length.

variable

In programming, a named storage location capable of containing a certain type of data that can be modified during program execution. See also data structure, data type.

variant

- 1 In OLE Automation, an instance of the VARIANT datatype, which can represent values of many different types, such as integers, floats, booleans, strings, pointers, etc.
- 2 In general, one of two or more data elements, functions, libraries, or whatever, exhibiting (usually slight) differences.

VBX

A custom control that can be used in Visual Basic as well as other Microsoft programming systems, including Visual C++. VBX controls are 16-bit dynamic-link libraries and are not supported in 32-bit applications. Visual Basic control files have a .VBX filename extension.

vector font

A scalable font in which the characters are drawn in arrangements of line segments rather than arrangements of curves or bits. Vector fonts are often used in applications that are optimized for output to plotters instead of printers. See also raster font, TrueType font.

Veronica

A database of the names of almost every menu item on thousands of gopher servers. This database can be searched from most major gopher menus. See also Archie.

version information

An application's company and product identification, product release number, and copyright and trademark notification. Version information is held in a standard form in the executable (.EXE) file or dynamic-link library (DLL) file and is accessible by various tools and Windows functions.

version resource

A resource that contains either text or binary data about an application's version information.

Very Easy Rodent-Oriented Net-wide Index to Computerized Archives

A database of the names of almost every menu item on thousands of gopher servers. This database can be searched from most major gopher menus. See also Archie.

view

- 1 (noun) A window object through which a user interacts with a document.
- 2 Or aspect. (noun) Generally, the manner in which data or a graphical image is displayed.

3 (verb) To display information on a computer screen, as in to view a file.

viewer

Program for opening image files and files in other special formats, such as audio and video.

viewport

A rectangle in device space that is used to specify a transformation between page and device space. The viewport extents (height and width) are always measured in pixels for a video display or in dots for printers.

viewport origin

The corner of the viewport from which the height and width of the viewport are measured.

VIRTKEY

In an accelerator table resource, a flag used to indicate that the keystroke value in the definition is a virtual key code.

virtual base class

A base class whose derived classes contain only one shared instance of its members even if the indirectly derived classes have done multiple inheritance. The class declaration for the derived class must have the keyword **virtual** before the base class specifier in order for it to be a virtual base class. In the following example, class `MultiC` has only one subobject of class `Base` even though it has inherited from more than one class derived from `Base`:

```
class Base { };
class DerivedA: virtual public Base { };
class DerivedB: virtual public Base { };
class MultiC: public DerivedA, public DerivedB { };
```

virtual destructor

A destructor implemented by declaring a base class's destructor with the keyword `virtual`. A virtual destructor ensures that, when `delete` is applied to a base class pointer or reference, it calls the destructor implemented in the derived class, if an implementation exists.

virtual device driver

A low-level software component that manages a single resource, such as a display screen or a serial port, on behalf of all running processes. A VxD is always 32-bit protected mode code and is frequently written in assembly language.

virtual function

A member function of a base class, where the function is declared with the keyword `virtual`. If a base class contains a virtual function and a derived class defines the same function, the function from the derived class is invoked for objects of the derived class, even if it is called using a pointer or reference to the base class.

virtual inheritance

In C++, a method of deriving classes by preceding the name of the base class with the keyword **virtual**. With virtual inheritance, indirectly derived classes contain only one shared instance of the base class's members even if they have done multiple inheritance. See also virtual base class.

virtual key code

In Windows, a device-independent value translated from the scan code by the keyboard device driver. The code identifies the keyboard key. See also VIRTKEY.

virtual memory

A memory-management scheme that allows an application to see a large, continuous block of primary memory (RAM) that, in reality, is a much smaller block of primary memory supplemented by secondary memory (such as a hard disk), with blocks of data being moved into and out of primary memory. See also swap file.

Virtual Reality Modeling Language

A vector-based language for modeling three-dimensional environments. It sends ASCII text files over the Internet, which are translated by the VRML viewing engine at the other end. VRML complements HTML.

visible region

In Windows, the portion of a window visible to the user.

Visual Basic control

A custom control that can be used in Visual Basic as well as other Microsoft programming systems, including Visual C++. VBX controls are 16-bit dynamic-link libraries and are not supported in 32-bit applications. Visual Basic control files have a .VBX filename extension.

void pointer

A pointer to an object of unknown type. See also typed pointer.

volatile

- 1 In C/C++ and J++, a type qualifier that declares an item whose value can legitimately be changed by something beyond the control of the program in which it appears, such as a concurrently executing thread.
- 2 More generally, describes information that is stored in memory and not preserved when a process or application terminates.

VRML

A vector-based language for modeling three-dimensional environments. It sends ASCII text files over the Internet, which are translated by the VRML viewing engine at the other end. VRML complements HTML.

VxD

A low-level software component that manages a single resource, such as a display screen or a serial port, on behalf of all running processes. A VxD is always 32-bit protected mode code and is frequently written in assembly language.

Glossary W

watch expression

An expression that can be evaluated as the program progresses and can be viewed in the Watch window during debugging.

wave file

A Microsoft standard file format for storing waveform audio data. Wave files have a .WAV filename extension.

waveform audio

A technique of re-creating an audio waveform from digital samples of the waveform.

weak external

A special type of external reference that allows the linker to fix up an address with an alternate symbol. By using a weak external record, a library developer can provide a method for the linker to use an alternate symbol if the code does not make any other references to the module in which the primary symbol resides.

Web browser

Software that renders Web pages on a local computer. See also World Wide Web.

what you see is what you get

A display method (pronounced WI-zy-wig) that shows documents and graphics as they will appear when printed.

white space

- 1 Spaces, tabs, and blank lines. White space sometimes refers to the C/C++ language-specific character constants that create white spaces; namely, ' ', '\t', and '\n'.
- 2 The empty areas in a window or the margins on a document.

wide character

Or Unicode character. A 2-byte multilingual character code. Any character, including technical symbols and special publishing characters, can be represented as a wide character. Note that a multibyte character or a double-byte character can be 2 bytes in size, but neither is a wide, or Unicode, character. See also wide-character constant.

wide-character constant

A 16-bit character constant that can specify a member of the extended execution character set. Wide-character constants can be used to express characters in alphabets that are too large to be represented by type **char**. Use wide-character constants in place of multicharacter constants to ensure portability. Syntactically, a wide-character constant is a character constant prefixed by the letter **L**.

WIN.INI

A text file that stores initialization information for Windows and Windows-based applications. The

WIN.INI file is configured with a number of sections and entries, depending on a particular system's hardware and software requirements. For example, entries in the [Desktop] section control the appearance of the desktop and the position of windows and icons.

Win32 platform

A platform that supports the Win32 API. These platforms include Intel Win32s, Windows NT, Windows 95, MIPS Windows NT, DEC Alpha Windows NT, and Power PC Windows NT.

window characteristic

An attribute of a window, such as size, position, or the presence of scroll bars.

window class

A set of attributes that Microsoft Windows uses as a template to create a window in an application. Windows requires that an application supply a class name, the window-procedure address, and an instance handle. Other elements may be used to define default attributes for windows of the class, such as the shape of the cursor and the content of the menu for the window.

window extent

The width (x-extent) or height (y-extent) of a window.

window handle

In the Win32 API, a 32-bit value (assigned by Windows) that uniquely identifies a window. An application uses this handle to direct the actions of functions to the window. A window handle has the **HWND** data type; an application must use this type when declaring a variable that holds a window handle.

window origin

- 1 The upper-left corner of a window's client area.
- 2 The corner of the window from which the extents are measured. See also viewport origin.

window procedure

A function, called by the operating system, that controls the appearance and behavior of its associated windows. The procedure receives and processes all messages to these windows.

window rectangle

The coordinate rectangle that encloses the entire window. The window rectangle is defined by two ordered pairs of numbers that define the upper-left and lower-right corners of the rectangle. See also client area.

window style

A named constant that defines an aspect of the window's appearance and behavior not specified by the window's class.

window-management function

A function that carries out an action on a window. For example **SetWindowPos** is a window-management function that can change the size, position, or Z order of a window.

Windows initialization file

A text file that stores initialization information for Windows and Windows-based applications. The WIN.INI file is configured with a number of sections and entries, depending on a particular system's hardware and software requirements. For example, entries in the [Desktop] section control the appearance of the desktop and the position of windows and icons.

Windows Open Services Architecture

A specification for an open set of API functions that allow Windows-based desktop applications to connect to multiple computing environments, with a single version of each application able to work with multiple platforms. With WOSA, applications can access available information regardless of the type of network in use, the types of computers involved, or the types of back-end services available.

Windows Write file

A document file that is associated with the Windows Write text editor. The default filename extension for Windows Write files is .WRI.

wizard

A special form of user assistance that guides the user through a difficult or complex task within an application. For example, a database program can use wizards to generate reports and forms. In Visual C++, the AppWizard generates a skeleton program for a new C++ application.

word alignment

Data in memory that is aligned on word boundaries. In 32-bit systems, word alignment means that the first byte of a data object is located at an address that is a multiple of 4.

word boundary

A memory address that is a multiple of the machine's word size, in bytes. See also word alignment.

wordwrap

To break lines automatically in order to keep the text within the boundaries of a window or the margins of a document.

worker thread

A thread that handles background tasks while the user continues to use an application. Tasks such as recalculation and background printing are examples of worker threads. See also user-interface thread.

working set

1 The set of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault. If free memory in the computer is above a threshold, pages are left in the working set of a process even if they are not in use. When free memory falls below a threshold, unneeded pages are trimmed from working sets.

2 The average amount of memory, either physical or virtual, used by the process. The longer a process is running, the more accurate this value is.

workspace

A window or task-management technique that consists of a container holding a set of objects, where the windows of the contained objects are constrained to a parent window. A workspace is similar to

the multiple document interface, except that the windows displayed within the parent window are of objects that are actually contained in the workspace.

workspace configuration file

A file created and maintained by the Developer Studio. The file contains all the information about the project workspace, including a list of all the projects, their associated source files, each project's output file, the settings and tools required to build each project, as well as the physical layout and characteristics of Developer Studio: window layout and characteristics, syntax coloring, editor preferences, and so on.

workspace window

In the Microsoft Developers Studio, a window that displays graphical representations of a project workspace. The panes in the window display projects and files, classes for C++ files, and the table of contents for the online help system.

World Wide Web

Or the Web. The portion of the global Internet that uses hypertext links to connect pages and resources in a way that lets you reach any page from any other page. Web browsers enable you to navigate the Web.

WOSA

A specification for an open set of API functions that allow Windows-based desktop applications to connect to multiple computing environments, with a single version of each application able to work with multiple platforms. With WOSA, applications can access available information regardless of the type of network in use, the types of computers involved, or the types of back-end services available.

wrapper class

A C++ class whose function is to provide an alternative interface for objects of another class. For example, C exceptions can be handled as typed exceptions, with the C++ catch handler, by using a C exception wrapper class.

wrapper function

A function whose purpose is to provide an interface to another function. Such an interface might create type safety where none existed in the original function.

write-only

An attribute of an object (a C++ ostream object, for example), a pipe, or a file specifying that data can only be written to it, not read from it.

WWW

Or the Web. The portion of the global Internet that uses hypertext links to connect pages and resources in a way that lets you reach any page from any other page. Web browsers enable you to navigate the Web.

WYSIWYG

A display method (pronounced WI-zy-wig) that shows documents and graphics as they will appear when printed.

Glossary X

x-extent

The width of a window. See also y-extent.

Glossary Y

y-extent

The height of a window. See also x-extent.

Glossary Z

Z order

Indicates a window's position in a stack of overlapping windows. This window stack is oriented along an imaginary axis, the z-axis, extending outward from the screen. The window at the top of the Z order overlaps all other windows. The window at the bottom of the Z order is overlapped by all other windows. An application sets a window's position in the Z order by placing it behind a given window or at the top or bottom of the stack.

zooming

Enlarging a selected window or portion of a graphical image in order to see finer details in the image.

& (ampersand)

- 1** Bitwise-AND operator. Two ampersands (**&&**) denote the logical-AND operator.
- 2** In C++, the reference operator, as in `void AddIt(int&x)`. This indicates that the address of `x` is used directly, and changes in value affect the original variable. Java does not allow reference operators. See also parameter passing.
- 3** In C/C++, the address-of operator, as in `pPtr = &nArray[2]`. Java does not allow address-of operators.
- 4** Indicates special characters in HTML code.

.APS

A binary version of the current resource file that is created by the Microsoft Developer Studio and used for quick loading of resources. Microsoft Developer Studio gives this file an .APS filename extension.

.BAT

An unformatted text file that contains one or more commands, either internal operating-system commands or program names. A batch file is executable and can be run from the command line.

.BMP

A file that contains a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

Bitmap files usually have a .BMP filename extension. See also bitmap, device-independent bitmap file.

.BSC

A file created from source browser information (.SBR) files, using the Microsoft Browse Information File Maintenance Utility (BSCMAKE). Browse information files can be examined in browse windows and usually have a .BSC extension.

.bss

A predefined data section of an executable file that contains uninitialized data, including all variables declared as static within a function or source module. The linker combines all the .bss sections in the object (.OBJ) and library (.LIB) files into one .bss section in the executable file.

.C

A text file containing C language code.

.CLW

A file that ClassWizard generates, containing information needed to edit existing classes or add new classes to a project. ClassWizard also uses the ClassWizard file to store information needed to create and edit message maps and dialog data maps, and to create prototype member functions.

ClassWizard files have a .CLW filename extension.

.COM

An executable binary (program) file whose code is limited to a single 64-kilobyte segment. Compact executable files usually have a .COM filename extension and are often used for utility programs and short routines. See also executable file.

.CPP

Or .CXX file. A text file containing C++ source code.

.CUR

A file that contains an image that defines the shape of a cursor on the screen. Cursor resource files usually have a .CUR filename extension.

.DEF

A text file that contains one or more statements describing various attributes of an executable module. Module-definition files usually have a .DEF filename extension. See also [dynamic-link library](#) (.DLL) file.

.DIB

A file containing an array of bits combined with several structures that specify the width and height of the bitmapped image (in pixels), the color format of the device where the image was created, and the resolution of the device used to create that image. The DIB file format ensures that bitmap graphics created in one application can be loaded and displayed in another application exactly the way they appear in the originating application. See also bitmap, bitmap file.

.DLG

A file that contains dialog-box source code. Note that a dialog file is not required for a Visual C++ project because Visual C++ keeps this code in the resource-definition file.

.DLL

A file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. In Win32, the operating system maps the dynamic-link libraries (DLLs) into the address space of a process when the process is starting up or while it is running. The process then executes functions in the DLL. Dynamic-link library files usually have a .DLL filename extension.

.EXE

A program file created from one or more source code files translated into machine code and linked together. The MS-DOS, Windows, and Windows NT operating systems use the .EXE filename extension to indicate that the file is a runnable program.

.EXP

A file that contains information about exported functions and data items. The Microsoft 32-Bit Library Manager tool (LIB.EXE) generates the exports file from the module-definition (.DEF) file. The linker uses the exports file to build the dynamic-link library (.DLL) file. Exports files have a .EXP filename extension.

.H

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

.HLP

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

.HM

A file that defines Help context IDs corresponding to the IDs of dialog boxes, menu commands, and other resources in an application. The AppWizard file MAKEHELP.BAT calls the MAKEHM tool to generate this file from the contents of a RESOURCE.H file. The Help map file has a .HM filename extension. See also Help project file.

.HPJ

A project file that controls how the Windows Help Compiler creates a Help (.HLP) file from topic files. The Microsoft Help Workshop is used to create a Help project file. The filename extension of a Help project file is .HPJ.

.HPP

Or .HXX file. An external source file, identified at the beginning of a C++ program, that contains commonly used data types and variables used by functions in a program. The #include directive is used to tell the compiler to insert the contents of a header file into the program.

.ICO

In Windows, a file that contains a bitmap of an icon. Icon files usually have a .ICO filename extension.

.ILK

A state file generated to hold status information for later incremental links of the program. The file has the same base name as the executable file or dynamic-link library and the filename extension .ILK. The incremental status file is created the first time the Incremental Linker (LINK.EXE) runs in incremental mode. LINK updates the file during subsequent incremental builds. LINK is the only tool that uses the .ILK file. See also incremental link.

.INI

In Windows, a file that an application uses to store information that otherwise would be lost when the application closes. Initialization files typically contain information such as user preferences for the configuration of the application. Initialization files usually have a .INI filename extension.

.LIB

A Common Object File Format (COFF) file generated by the Microsoft 32-bit library manager tool, LIB, for standard and import libraries. The default filename extension for these files is .LIB. See also dynamic-link library (.DLL), static-link library.

.MAK

A file that contains all commands, macro definitions, options, and so on to specify how to build the projects in a project workspace. A makefile has the filename extension .MAK and usually has the same base name as the workspace configuration (.MDP) file.

.MAP

A text file that contains information about the program being linked, including the groups in the program and a list of public symbols. The linker names the mapfile with the base name of the program and the filename extension .MAP.

.MDP

A file created and maintained by the Developer Studio. The file contains all the information about the project workspace, including a list of all the projects, their associated source files, each project's output file, the settings and tools required to build each project, as well as the physical layout and characteristics of Developer Studio: window layout and characteristics, syntax coloring, editor preferences, and so on.

.OBJ

A file containing object code and/or data generated by a compiler or an assembler from the source code of a program. Object files generated by the Visual C++ compiler have a .OBJ filename extension. See also Common Object File Format (COFF).

.ODL

In OLE Automation, text files containing a description of an application's interface. Object description language scripts are compiled into type libraries using the MkTypLib tool included with the OLE Software Development Kit.

.OLB

A dynamic-link library with a type library resource. An object library file typically has a .OLB filename extension.

.PBI

In a profiling operation, a file that provides condensed information to the Visual C++ profiler (PROFILE). The PREP program generates a profiler batch input file the first time the profiler is run on a program. The default filename extension for profiler batch input files is .PBI. See also profiler batch output file, profiler batch text file.

.PBO

An intermediate file generated by the Visual C++ profiler (PROFILE) and used to transfer information between profiling steps. See also profiler, profiler batch input file, profiler batch text file.

.PBT

In a profiling operation, the file generated by the PREP program and used as input to the PLIST program to generate a human-readable profile of the source code. See also profiler, profiler batch input file, profiler batch output file.

.PCH

A file containing compiled code for a portion of a project. Subsequent builds combine this file with the uncompiled code, thus shortening the overall compile time. The default filename extension for a precompiled header file is .PCH.

.PDB

A file used by the build tools to store information about a user's program. The program database file speeds linking during the debugging phase of development by keeping the debugging information separate from the object files.

.RC

Or resource script file. A text file containing descriptions of resources from which the resource compiler creates a binary resource file. For Microsoft Windows applications, resource-definition files usually have a .RC filename extension. For Apple Macintosh applications, such files are typically named with a .R extension and written with the Apple Rez script language. See also compiled resource (.RES) file.

.REG

In OLE applications, a text file description of the classes supported by a server application. When a server application is installed in a system, the contents of its registration entry file are merged with the system registry. Registration entry files usually have a .REG filename extension.

.RES

Or binary resource file. A binary file that contains a Windows-based application's resource data and is created by the resource compiler from the resource-definition (.RC) file. Compiled resource files usually have a .RES filename extension. See also Macintosh binary resource file, resource compiler.

.RSC

A Macintosh resource file that has been created from a Windows resource script (.RC) and compiled using the Windows Portability Library version of the Windows Resource Compiler (RC.EXE). By default, .RSC is the filename extension for Macintosh binary resource files.

.RTF

A file that contains encoded, formatted text and graphics for easy transfer between applications. The rich-text encoding format is commonly used by document-processing programs such as Microsoft Word for Windows and for generating online Help files. Rich-text format files usually have a .RTF filename extension.

.SBR file

An intermediate file that the compiler creates for use by the Microsoft Browse Information Maintenance Utility (BSCMAKE). There is one .SBR file for each object (.OBJ) file. BSCMAKE uses the .SBR files to create a browse information (.BSC) file.

.TLB

Or OLE library. An OLE compound document file containing standard descriptions of data types, modules, and interfaces that can be used to fully expose objects for OLE Automation. The type library file usually has a .TLB filename extension and can be used by other applications to get information about the automation server.

.TXT

A human-readable file composed of text characters. A text file is usually identified by a file extension of .TXT. See also binary file, rich-text format file.

.WAV

A Microsoft standard file format for storing waveform audio data. Wave files have a .WAV filename extension.

.WRI

A document file that is associated with the Windows Write text editor. The default filename extension for Windows Write files is .WRI

@ (at sign)

- 1** In e-mail addresses, separates the user name from the domain name. See also address.
- 2** Points to the input command file used by CL or LINK, as in LINK @LINK.RSP. See also command file.

16-bit application

A program written for a system (such as MS-DOS or Microsoft Windows version 3.1) that uses a 16-bit segmented architecture, in which each memory address points to a 16-bit word.

16-bit character

A character that is 2 bytes in size, unlike an ANSI character, which is 1 byte in size. Sixteen-bit characters are found in Unicode sets, multibyte character sets (MBCS), and double-byte character sets (DBCS).

32-bit application

A program written for a system (such as Microsoft Windows NT or Windows 95) that uses a 32-bit architecture, in which each memory address points to a 32-bit word.

8.3 filename convention

The naming convention for filenames in MS-DOS that allows up to eight characters, with an optional period and three-character extension. See also base name, filename extension.

A5-relative reference

An address designated by a 16-bit offset from the A5 register in the 680x0-based Apple Macintosh.
An A5-relative reference is similar to the near data address in Intel-based machines.

ABC character spacing

See ABC width.

ABC width

Or ABC character spacing. The amount of spacing required for a single glyph in a font. A spacing is added to the current position before drawing the glyph. B spacing is the width of the black part of the glyph. C spacing is added to the current position to account for the white space to the right of the glyph. The total advanced width is given by $A+B+C$. See also overhang, underhang.

abnormal termination

In exception handling, the condition that occurs when a program leaves the **try** block of a **try-finally** statement before the code executes to the closing brace. For example, statements such as **return**, **goto**, **continue**, and **break** can cause abnormal termination.

absolute symbol

A symbol that contains a constant value for an address in memory not associated with a program address.

absolute time

An expression of time that stays the same regardless of the time zone.

abstract base class

See abstract class.

abstract class

Or abstract base class. In C++, a class that cannot be instantiated but is used as a base from which other classes can be derived. An abstract class contains at least one pure virtual function; if its derived classes do not implement these pure virtual functions, then they, too, become abstract classes.

accelerator key

Or keyboard accelerator, shortcut key, keyboard shortcut. A keystroke or combination of keystrokes that invokes a particular command. See also accelerator table, access key.

accelerator resource

See accelerator table.

accelerator table

Or accelerator resource. A data structure that contains a list of accelerator keys and the command identifiers associated with them.

access key

Or mnemonic key. The key that corresponds to an underlined letter on a menu or dialog-box item. An access key can be used to invoke the command associated with that menu item or dialog-box item. See also accelerator key.

access mode

A mode that determines the manner in which entry to a file, directory, or other data source object is permitted, as well as the manner in which changes are made to the data. See also direct mode, transacted mode.

access privileges

In object-oriented programming, the degree to which a class grants outside access to its data and functions. See also friend.

access token

A group of security attributes permanently attached to a process when a user logs on to the operating system. An access token contains privileges and security identifiers for a user, global group, or local group. The privileges regulate the use of some system services and the security identifiers regulate access to objects that are protected by access-control lists (ACLs). There are two kinds of access tokens: primary and impersonation. See also impersonation token, primary token, privilege, security identifier (SID).

access-control list

A list of security protections that applies to an object (a file, process, event, or anything else having a security descriptor). An entry in an ACL is an access-control entry (ACE). There are two types of access-control lists: discretionary and system. See also security descriptor.

ACL

A list of security protections that applies to an object (a file, process, event, or anything else having a security descriptor). An entry in an ACL is an access-control entry (ACE). There are two types of access-control lists: discretionary and system. See also security descriptor.

activation

The process of making a window or other object operational. Activation of a window, for example, changes the color of the title bar, moves the window to the top of the Z order, and enables keyboard focus. In OLE, the term activation is often used to refer to in-place activation of embedded objects.

active HTML document

A Web page that contains ActiveX controls, active scripts, or Java applets.

active state

- 1 In OLE, the state of an OLE object in a compound document when the user can edit the embedded object without leaving the container document's window.
- 2 In general, the state of a program, document, device, or portion of the screen that is currently operational.

ActiveX

All component technologies built on Microsoft's Component Object Model (COM), other than Object Linking and Embedding.

ActiveX control

The new name for programmable elements formerly known variously as OLE Controls, OCXs, or OLE Custom Controls. Controls previously built with the MFC Control Developer's Kit meet the ActiveX control specification.

ActiveX document

A document that contains ActiveX controls, Java applets, or ActiveX document objects. Also called active object or active script.

ActiveX scripting

Microsoft technology for connecting third-party script engines to applications.

ActiveX server extension

A dynamic-link library (DLL) that creates server extensions on any ISAPI-compliant Web server.

ActiveX server filter

A dynamic-link library (DLL) that intercepts and processes notifications directed to any ISAPI-compliant Web server.

ActiveX server framework

Microsoft tools for creating extensions to ISAPI-compliant Web server software. See also ActiveX server extension, ActiveX server filter, ActiveX scripting.

actual argument

- 1 Or actual parameter. A unit of information (variable, constant, pointer, etc.) passed in a macro call or inside the parentheses of a function call that the called function then converts to its formal parameter.
- 2 In the Visual C++ documentation, the argument passed to a C++ template class.

actual parameter

See actual argument.

actual parameter list

The arguments specified in a particular method or function call. See also formal parameter list.

address

The path to a value, object, document, page, or other destination. An address can be a memory location, a URL (address to an Internet site) or a UNC network path (address to a file on a local area network). An address may also contain more specific information such as a database object, a bookmark in a file, or a spreadsheet cell range to which the main address points.

address space

- 1** Or memory space. The portion of memory allocated to a given process. See also heap.
- 2** Or memory space. More generally, the range of memory locations to which a microprocessor can refer. Effectively, a computer's address space is the amount of memory a microprocessor could use if all the memory was available.

address-of (&) operator

In C/C++, a unary operator that gives the address of its operand, which can be either a function name or an l-value. The result of the address-of operation is a pointer to the operand. The type addressed by the pointer is the type of the operand. Note that the ampersand (&) character is also used by the bitwise-AND (&) operator. See also bitwise operator, indirection (*) operator.

AFXDLL

Redistributable dynamic-link libraries (DLLs) containing the entire 32-bit Microsoft Foundation Class Library. AFXDLL enables you to build an application without statically linking to the MFC object-code libraries. The architecture is useful in building MFC extension DLLs and in sharing the class library between multiple executable files, thus saving disk space and memory.

aggregate object

In the Component Object Model, an object whose implementation of certain interfaces is provided by one or more of its contained objects.

aggregate type

Or complex type, nonscalar type. In C/C++, a structure, union, or array data type. In C++, a class can also be an aggregate type, provided it does not have constructors, nonpublic members, base classes, or virtual functions.

aliasing

- 1** In programming, to use an alternate name to refer to a memory location that is already referred to by a different name, or to allow two pointers to point to the same memory location. The alternate name is an alias.
- 2** In computer graphics, a rendering technique that assigns to pixels the color of the primitive being rendered, regardless of whether that primitive covers all or only a portion of the pixel's area. This results in a jagged, or stairstep, appearance of certain design elements, such as diagonal lines, curves, and circles.
- 3** More generally, to substitute one name, or alias, for another name, or for a group of names. For example, a long Macintosh filename could be aliased with an MS-DOS 8.3-format filename.

allocation

See memory allocation.

alpha value

In mixing models, the component used to control color blending. See also red, green, blue, alpha (RGBA).

ambiguity

- 1** In derived classes, a condition that occurs when an expression can refer to more than one data type, object, or function. Use of the scope resolution (::) operator resolves the ambiguity.
- 2** In syntax, a condition that can occur with a type cast, especially a function-style type cast. For example, it is unclear whether the expression `char *aName(String(s))` is a function declaration or an object declaration with a function-style cast as the initializer.
- 3** More generally, two or more meanings for a single expression, which is anathema to a binary system.

ambiguous expression

- 1** An expression whose value depends on a particular order of evaluation where the language does not define one. For example, the values received by `func(i, ++i)` depend on whether its parameters are passed from right to left or from left to right.
- 2** An expression that cannot be evaluated because the types used in the expression are not unique.

American National Standards Institute

An organization of American industry and business groups dedicated to the development of trade and communication standards. ANSI sets standards for C and other programming languages to eliminate variations that could cause problems in transporting a program from one type of computer system or environment to another.

American Standard Code for Information Interchange

The dominant standard for coding information on computers and related equipment. The ASCII coding scheme assigns numeric values to letters, numbers, punctuation marks, and certain other characters, enabling computers and computer programs to exchange information. See also ANSI character set, Unicode.

anchor

The HTML element that connects Web documents. Anchors either jump to another location, or are jumped to by other anchors. They are similar in function to bookmarks.

anonymous union

In C++, a union without a tag or declarators. An anonymous union declares an unnamed object and cannot have member functions or private or protected members.

ANSI

An organization of American industry and business groups dedicated to the development of trade and communication standards. ANSI sets standards for C and other programming languages to eliminate variations that could cause problems in transporting a program from one type of computer system or environment to another.

ANSI c

The American National Standards Institute's 1990 version of the C language, which specifies the formal syntax and semantics of the language, the standard libraries that are included, and the way in which the compiler is to translate the program. ANSI compliance promotes portability of programs across different platforms.

ANSI character set

An 8-bit character set that contains the 7-bit ASCII standard character set as well as currency and mathematical symbols, accented characters, and other characters not normally found on the keyboard. Microsoft Windows version 3.1 and its applications use the ANSI character set internally. See also OEM character set, Unicode.

ANSI string

A string composed of characters from the ANSI character set.

API

A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system. For computers running a graphical user interface, an API manages an application's windows, icons, menus, and dialog boxes.

App Studio precompiled file

A binary version of the current resource file that is created by the Microsoft Developer Studio and used for quick loading of resources. Microsoft Developer Studio gives this file an .APS filename extension.

Apple Shared Library Manager

A library of code and resources that can be used simultaneously by more than one Macintosh application. (ASLM is not supported on the Power Macintosh.) See also stand-alone code.

applet

- 1** An HTML-based program that the browser temporarily downloads to a user's hard drive, where it runs when the Web page is open. Most often, applet refers to programs developed with Java.
- 2** Any small application.

AppleTalk

A simple local area network developed by Apple Computer that can be used by both Apple and non-Apple computers for communicating and sharing resources such as printers and file servers.

application class

The class, derived from the MFC class **CWinApp**, that encapsulates the initialization, running, and termination of a Windows-based application. An application must have exactly one object of an application class. See also application object.

application framework

Or framework. A group of C++ classes in the Microsoft Foundation Class Library that provides the essential components of an application for Windows. The application framework defines the skeleton, or framework, of an application and supplies standard user-interface implementations that can be placed onto the skeleton. See also class library.

application object

The single instance of the application class. The application object controls documents, views, frame windows, and templates, and specifies application behavior such as initialization and cleanup for every instance of the application.

application programming interface

A set of routines that an application uses to request and carry out lower-level services performed by a computer's operating system. For computers running a graphical user interface, an API manages an application's windows, icons, menus, and dialog boxes.

application queue

In Microsoft Windows, a repository for messages awaiting processing by a particular application. As Windows removes messages from its system message queue, it dispatches the messages that contain application input to the relevant application queue. See also message loop, message queue.

Archie

A tool for finding files stored on anonymous FTP sites. You need to know the exact file name or a substring of it. See also Very Easy Rodent-Oriented Net-wide Index to Computerized Archives (Veronica).

archive

- 1 An object, derived from the MFC class **CArchive**, that provides a type-safe buffering mechanism for writing or reading serializable objects to or from a **CFile** object—a disk file or a memory file (perhaps representing the Clipboard). A given **CArchive** object either stores (writes, serializes) data or loads (reads, deserializes) data, but never both. See also serialization.
- 2 In general, a stored backup copy of data or a program, or the act of storing a backup copy of data or a program.

argument

A value or an expression used with an operator or passed to a subprogram (subroutine, procedure, or function). The program then carries out operations using the arguments. Synonymous with parameter.

arithmetic shift

A shift operation whose left operand is a signed quantity. In a right-shift operation, the sign bit is propagated into the bits vacated by the shift. See also logical shift.

array

An aggregate data type in which all the data items (elements) are of the same type. The elements are accessed by specifying the name of the array and a subscript (index), which represents the element's offset from the base address of the array. For example, `myArray[3]` represents the fourth element in the array. See also subscript (`[]`) operator.

ASCII

The dominant standard for coding information on computers and related equipment. The ASCII coding scheme assigns numeric values to letters, numbers, punctuation marks, and certain other characters, enabling computers and computer programs to exchange information. See also ANSI character set, Unicode.

ASCII character

A character that is part of the American Standard Code for Information Interchange (ASCII). The ASCII character set is a standard 7-bit code for representing characters—letters, digits, punctuation, and control instructions—with binary values.

ASLM

A library of code and resources that can be used simultaneously by more than one Macintosh application. (ASLM is not supported on the Power Macintosh.) See also stand-alone code.

aspect ratio

The ratio of a pixel's width to height on a particular device. Information about a device's aspect ratio is used in the creation, selection, and display of fonts.

assertion

A Boolean statement in the debug version of a program that tests a condition that should, if the program is operating correctly, evaluate as true. If the condition is false, an error has occurred and the program will typically issue an error message that gives the user the option to abort the program, activate the debugger, or ignore the error.

assignment operator

An operator used to assign a value to a variable or a data structure. An example of an assignment operator is the simple assignment (=) operator, which assigns the value of its right operand to its left operand.

assignment statement

A statement that assigns a value to a variable or data structure. An assignment statement is usually composed of a destination variable, an assignment operator, and an expression to be assigned.

associativity

The order (either right to left or left to right) in which expressions are evaluated when adjacent operators have equal precedence and subexpressions are not enclosed in parentheses. For example, the expression $10 < 20 < 5$ evaluates to 1 (true) because the less-than (<) relational operator has left-to-right associativity, causing the expression to be evaluated $(10 < 20) < 5$.

asynchronous operation

1 Or overlapped I/O. In programming for Windows, a task that proceeds in the background, allowing the thread that requested the task to continue to perform other tasks. See also synchronous operation.

2 More generally, an operation that proceeds independently of any timing mechanism such as a clock.

asynchronous processing

In ODBC, a method of processing transactions in which the database driver returns control to an application before a function call completes; the application can continue nondatabase processing while the driver completes the function in progress. See also synchronous processing.

atomic operation

An operation that never has its execution suspended while partially completed. See also synchronous operation.

audio-video interleaved

A Microsoft technology for displaying full-motion video in a window. An AVI file interleaves waveform audio and video data and has a .AVI filename extension.

authentication

The process of determining the identity of a user attempting to access a system. Note that the user can be a person, a computer, or a process.

authentication token

A portable device used for authenticating a user. Authentication tokens operate by challenge/response, time-based code sequences, or other techniques.

auto storage class

See automatic storage class.

automatic storage class

Or **auto** storage class. In C++, the storage class for objects and variables that are local to the block of code where they are declared. The automatic storage class can be declared explicitly, by using the keywords **auto** or register, or implicitly, by default. See also local variable, static storage class.

automation client

In OLE Automation, an application that can manipulate exposed objects belonging to another application (called the OLE Automation server). The client drives the server application by accessing the objects' properties and functions. Microsoft Visual Basic is an example of an OLE Automation client application. See also automation server.

AVI

A Microsoft technology for displaying full-motion video in a window. An AVI file interleaves waveform audio and video data and has a .AVI filename extension.

backbone

A high-performance network connecting other networks.

background color

The color of the client area of an empty window or display screen, on which all drawing and color display take place. See also background mode, foreground color.

background mode

A mode that defines how background colors are mixed with existing window or screen colors for bitmap and text operations. The background mode can be either opaque (the default) or transparent.

backward compatibility

- 1** Ensuring that existing applications will continue to work in the new environment.
- 2** Ensuring that the new release of an application will be able to handle files created by a previous version of the product.

bandwidth

The amount of data that can travel through a circuit, usually indicated in bits per second (bps). A measure of capacity, not speed.

bandwidth on demand

In ISDN, the ability to aggregate B channels as the data traffic exceeds preset thresholds.

base address

In relation to memory locations, the portion of a two-part address that remains constant and provides a reference point, or base, from which the location of a byte of data can be calculated. A base address is accompanied by an offset value that is added to the base to determine the exact location (the absolute address) of the information.

base class

In C++, a class from which other classes are derived by inheritance. See also abstract class, derived class, virtual base class.

base line

In the printing and on-screen display of characters, an invisible, horizontal line within a character cell of a given font. Most characters sit on the base line, although some characters, such as *g* and *j*, descend below it.

base name

The portion of the filename that precedes the extension. For example, SAMPLE is the base name of the file SAMPLE.ASM. See also filename extension.

base priority

In multitasking, the priority level that provides the basis for calculating a thread's dynamic priority for CPU time. A thread inherits its base priority from the process in which the thread was created. The scheduler, which determines the order in which threads should execute, can temporarily boost the priority of a thread, but it cannot reduce the priority below this base priority.

BASED_CODE

A 16-bit MFC macro that ensures that data will be placed in the code segment instead of the data segment. Under Win32, this macro expands to nothing and is provided for backward compatibility.

basic input-output system

A set of routines that work closely with the hardware and the operating system to support the transfer of information between elements of the system, such as memory, disks, and the monitor.

basic rate interface

An ISDN service that consists of two 64-Kbps bearer channels (B channels) and one 16-Kbps signaling channel (D channel) referred to as 2B+D.

batch file

An unformatted text file that contains one or more commands, either internal operating-system commands or program names. A batch file is executable and can be run from the command line.

big-endian

One of two byte-ordering conventions used on different machines. In big-endian addressing, the address points to the most significant byte of the word. Scalar Processor ARChitecture (SPARC) and Motorola 680x0 machines are big-endian machines. See also little-endian.

binary

- 1** The base-2 counting system, whose digits are 0 and 1.
- 2** An executable file—one that is in binary, or machine-readable, form. See also binary file.
- 3** A means of opening a file for input and output (binary mode versus text mode).

binary file

Or binaries. A file that consists of binary data or executable code. For example, a C++ executable file is a binary file. See also text file.

binary large object

- 1 Or binary data object. A large piece of data, such as a bitmap. A BLOB is characterized by large field values, an unpredictable table size, and data that is formless from the perspective of a program.
- 2 A keyword designating the BLOB structure, which contains information about a block of data.

binary operator

An operator that takes two operands. In C, binary operators are the multiplicative operators (*, /, %), additive operators (+, -), shift operators (<<, >>), relational operators (<, >, <=, >=, ==, !=), bitwise operators (&, |, ^), logical operators (&&, ||), the sequential-evaluation operator (,) the assignment operator (=), and the compound-assignment operators (+=, *=, etc.). See also ternary operator, unary operator.

binding

- 1** The association of two pieces of information with one another, most often used in terms of a symbol (such as the name of a variable) with some descriptive information (such as a memory address, a data type, or an actual value).
- 2** In networking, a process that establishes the initial communication channel between the protocol driver and the network adapter card driver.
- 3** In OLE, the process of getting a compound-document object into a running state so that it can be activated. See also activation.
- 4** A Windows Sockets keyword that identifies the **bind** socket library routine. The bind routine associates a local address with a socket.

BIOS

A set of routines that work closely with the hardware and the operating system to support the transfer of information between elements of the system, such as memory, disks, and the monitor.

bit field

A structure member whose width is specified in bits, or binary digits, rather than being specified implicitly as characteristic of the data type. When viewed in binary form, each bit or group of bits in the bit field corresponds to a specific field of information.

bitmap

- 1 Or pixel image, pixel map. An array of bits that contains data describing the colors found in a rectangular region on the screen (or the rectangular region found on a page of printed paper).
- 2 Or bit image. A sequential collection of bits that represents, in memory, an image to be displayed on the screen or printed. See also bitmap file.

bitmap file

A file that contains a collection of structures that specify or contain the following elements:

- A header that describes the resolution of the device on which the rectangle of pixels was created, the dimensions of the rectangle, the size of the array of bits, and so on.
- A logical palette.
- An array of bits that defines the relationship between pixels in the bitmapped image and entries in the logical palette.

Bitmap files usually have a .BMP filename extension. See also bitmap, device-independent bitmap file.

bitmap-stretching mode

A mode that defines how information is removed from or combined together in bitmaps that are stretched or compressed. For example, a particular bitmap-stretching mode may preserve black pixels at the expense of colored or white pixels.

bitwise operator

An operator that works on individual bits of its operands. The bitwise-NOT (~) operator produces the bitwise complement of its operand. For example, $\sim 1010 = 0101$. The binary operators—bitwise-AND (&), bitwise-OR (|), and bitwise-XOR (^)—compare each bit of the first operand to the corresponding bit of the second operand. For example, $1010 \& 0110 = 0010$. See also logical operator.

BLOB

- 1 Or binary data object. A large piece of data, such as a bitmap. A BLOB is characterized by large field values, an unpredictable table size, and data that is formless from the perspective of a program.
- 2 A keyword designating the BLOB structure, which contains information about a block of data.

BOND

In ISDN, the ability to aggregate B channels as the data traffic exceeds preset thresholds.

bookmark

A marker that uniquely identifies a specific record or row in a database, a specific line in source code, or an item or location in a word-processing file. The HTML equivalent of a bookmark is an anchor with the NAME attribute; this type of anchor is used as a destination for hyperlinks. When creating HTML documents in Word for Windows, for example, you use bookmarks to create anchors with the NAME attribute.

Boolean

A binary algebra that uses the logical operators AND, OR, XOR, and NOT, and whose outcomes consist of logical values (either TRUE or FALSE). The keyword boolean indicates that the expression or constant expression associated with the identifier takes the value TRUE or FALSE. See also conditional expression.

bounding rectangle

Or bounding box. A rectangular area that defines the outer limits of a rounded shape such as an ellipse, arc, or pie.

breakpoint

A location in a program where execution is stopped to allow the developer to examine the program's code, variables, and register values and, as necessary, to make changes, continue execution, or terminate execution.

BRI

An ISDN service that consists of two 64-Kbps bearer channels (B channels) and one 16-Kbps signaling channel (D channel) referred to as 2B+D.

bridge

1 A device that connects two segments or components of a network that use the same protocols. Used to overcome limitations on the physical length of a segment. See also brouter, router.

2 In ISDN, a hardware device used for forwarding data between two networks. A bridge reads each packet's address information, which is written into the packet at the data-link layer of the OSI model. Unlike a router, a bridge is not interested in the address on the remote LAN, but only in whether or not the packet should be forwarded.

brouter

A device combining the functions of a bridge and a router by connecting two segments of a network and connecting two networks or a network to the Internet. See also bridge, router.

browse buttons

- 1 In Windows Help, a pair of buttons used to browse backwards and forwards through a sequence of Help topics. A browse sequence typically consists of two or more related topics that are intended to be read sequentially.
- 2 In dialog boxes, buttons used to view a list of network servers, directories, files, etc.

browse information file

A file created from source browser information (.SBR) files, using the Microsoft Browse Information File Maintenance Utility (BSCMAKE). Browse information files can be examined in browse windows and usually have a .BSC extension.

browser

A program used to view formatted Web documents.

brush

A graphics object used to paint the interior of shapes and paths. A brush describes an 8- by 8-pixel bitmap. See also logical brush, physical brush.

BSCMAKE

The Microsoft Browse Information File Maintenance utility. BSCMAKE uses source browser information (.SBR) files created by the compiler using the /FR or /Fr option, which are used to create browse information (.BSC) files.

buffer

1 (noun) An intermediate repository of data—a reserved portion of memory in which data is temporarily held pending an opportunity to complete its transfer to or from a storage device or another location in memory.

2 (verb) To store or collect in a buffer.

build

1 (noun) The process of compiling and linking source code to generate an executable program, library, Help file, or other run-time file. The files produced from the build process are also sometimes referred to as a build.

2 (verb) To compile and link source code in order to generate an executable program, Help file, or other run-time file.

build tag

A string assigned to a topic that the Help project file can specify to include or not include in a **build**. Build tags can be made up of any alphanumeric characters and are not case-sensitive. They provide a means of supporting different versions of a Help system without the need to create different source files for each version. Topics without **build tags** are always included in a **build**, along with all tagged topics not expressly excluded from that particular **build** in the **build** expression.

bulk record field exchange

When bulk row fetching is implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), DAO record field exchange (DFX), and dialog data exchange (DDX).

bulk RFX

When bulk row fetching is implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), DAO record field exchange (DFX), and dialog data exchange (DDX).

bulk row fetching

In ODBC, the process of retrieving multiple rows from the data source in a single fetch operation. The number of rows retrieved depends on the recordset object's setting for the rowset size.

button control

A graphical control that enables a user to provide input to an application. Windows provides five kinds of button controls: pushbuttons, check boxes, radio buttons, group boxes, and owner-draw buttons.

byte

A unit of information consisting of 8 bits. A byte, or binary term, is the smallest collection of bits that can be accessed directly.

bytecode

Machine-independent code generated by the Java compiler and executed by the Java interpreter or compiled at the last minute by a JIT compiler.

C calling convention

The C standard for calling a function—that is, pushing arguments onto the stack from right to left (in reverse order from the way they appear in the argument list). After the function returns, the calling function removes the arguments from the stack. The C calling convention permits a variable number of arguments to be passed. See also calling convention.

C linkage specifier

A declaration of a function or object as `extern C`, indicating to the C++ compiler that the function name that follows the linkage specifier is an undecorated C function. C linkage allows existing C code to be used in new C++ applications. See also linkage specification.

C source file

A text file containing C language code.

C++ exception handling

Built-in support provided by the C++ language for handling anomalous situations, known as exceptions, that may occur during the execution of a program. With C++ exception handling, a program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control. See also structured exception handling (SEH).

C++ header file

Or .HXX file. An external source file, identified at the beginning of a C++ program, that contains commonly used data types and variables used by functions in a program. The #include directive is used to tell the compiler to insert the contents of a header file into the program.

C++ source file

Or .CXX file. A text file containing C++ source code.

cache

A special memory subsystem in which frequently used data values and instructions are duplicated for quick access.

call level interface

A library of function calls that support SQL statements and conform to the SQL Access Group Call Level Interface specification. These calls are typically used for dynamic access to data. ODBC is a call level interface.

call stack

An ordered list of functions that have been called but have not returned, with the currently executing function listed first. Each call is optionally shown with the arguments and types passed to it. During a debug session, you can view the functions that have been called but have not returned.

callback function

An application-defined function that a system or subsystem (Windows, for example) calls. Typically, this happens when an event occurs or when windows or fonts are being enumerated. Examples of callback functions include window procedures, dialog-box procedures, and hook procedures. Callback functions are also used to process dynamic data exchange (DDE) transactions.

calling convention

A convention that determines the order in which arguments passed to functions are pushed on the stack (the calling sequence), whether the calling or called function removes the arguments from the stack, and the name-decorating convention the compiler uses to identify individual functions. See also C calling convention, calling sequence.

calling sequence

Determines the order in which parameters are pushed onto the stack during a function call and which code block is responsible for the stack pointer. Typically, the C compiler generates code that pushes parameters on the stack from right to left, beginning with the last parameter. See also calling convention.

caret

- 1** Or insertion point. A flashing line, block, or bitmap that marks the location at which inserted text will appear in a window's client area.
- 2** (^) When preceding a single uppercase letter, indicates a control character. For example, ^C is the same as CTRL+C.
- 3** (^) A regular expression used to indicate either the beginning of a line or, when used within brackets, to indicate an exception.

carriage return character

A control character that tells a computer or printer to return to the beginning of the current line. This character can have a different textual representation on different platforms, but it always has the ASCII value of 13. See also carriage return–linefeed (CR-LF) pair.

carriage return–linefeed (CR-LF) pair

The combination of a carriage return character (ASCII 13) and a linefeed character (ASCII 10), represented in C/C++ by the newline (`\n`) character.

casting

Explicit or implicit conversion of one data type to another.

catastrophic error

See fatal error.

catch block

Or catch handler. In C++, a block of exception-handling code preceded by the keyword **catch**. The code in the catch block is executed only if the code in the try block throws an exception of the type specified in the **catch** statement. See also C++ exception handling, throw expression.

CGI

A mechanism that allows a Web server to run a program or script on the server and send the output to a Web browser. See also Internet Server Application Programming Interface (ISAPI).

Challenge Handshake Authentication Protocol

In ISDN, a type of signaling authentication that uses a pair of secret codes consisting of up to 16 characters. CHAP is shared by communications devices on both ends.

CHAP

In ISDN, a type of signaling authentication that uses a pair of secret codes consisting of up to 16 characters. CHAP is shared by communications devices on both ends.

character constant

A member of the source character set, the character set in which a program is written, surrounded by single quotation marks ('[.thsp]). Character constants are used to represent characters in the execution character set on the machine where the program executes.

character index

The number of characters from the beginning of an edit control.

character-mode application

An application that does not provide its own graphical user interface (GUI). The Win32 API provides consoles for managing input and output for character-mode applications. See also console application.

checksum

An error-detection scheme that involves creating a sum of the bits in a set of bytes of data and using that sum to later check for a change in the data. Checksums are commonly used in communications software to check for data transmission errors.

child control

A child window used in conjunction with another window (its parent) to carry out simple input and output (I/O) tasks.

child process

A process initiated by another process (the parent process). The child process can operate independently from the parent process. Further, the parent process can suspend or terminate without affecting the child process.

child window

A window that has the WS_CHILD or WS_CHILDWINDOW style and is confined to the client area of its parent window, which initiates and defines the child window. Typically, an application uses child windows to divide the client area of a parent window into functional areas. See also child control, sibling window.

chord

A closed figure bounded by the intersection of an ellipse and a line segment. In Windows, a chord is outlined by using the current pen and filled by using the current brush.

CL environment variable

An environment variable used to specify files and options for the compiler/linker so you do not have to specify them on the command line.

CL.EXE

Or CL. A driver program that controls the Microsoft C and C++ compilers and linker. The compilers produce Common Object File Format (COFF) object files. The linker produces executable (.EXE) files, dynamic-link libraries (DLLs), or static-link libraries.

class

A type that defines the interface of a particular kind of object. A class definition defines instance variables and methods, class variables and methods, and specifies the immediate superclass (or superclasses) and the interfaces that the class implements.

class declaration

In C++, the mechanism for declaring an aggregate data structure of type **class**. A class declaration provides a list of its members (such as functions, data, and other classes), specifies any friends of the class, and defines the level of visibility for all members.

class identifier

A universally unique identifier (UUID) that identifies a type of OLE object. Each type of OLE object (item) has its CLSID in the registry so that it can be loaded and programmed by other applications. For example, a spreadsheet may create worksheet items, chart items, and macrosheet items. Each of these item types has its own CLSID that uniquely identifies it to the system. See also registration entry file.

class library

A set of related C++ classes that can be used in an application, either as originally defined or as the source for other derived classes. The Microsoft Foundation Class Library included in Visual C++ is an example of a class library that defines a framework for integrating the user interface of an application for Windows with the rest of the application.

class method

In Java, any method that can be invoked using the name of a particular class. Since the declaration uses the keyword **static**, these are called static member functions in C++. Class methods, which are defined in class definitions, affect the class as a whole, not a particular instance of the class. Compare with instance method.

class scope

In C++, the degree of visibility afforded to a name (function or variable, for example) when it is declared within a class declaration. The name is accessible from outside the class by using the scope-resolution (::) operator. See also file scope, function scope, function-prototype scope, local scope.

class variable

In Java, a data item associated with a particular class as a whole, not with particular instances of the class. Class variables are defined in class definitions. Equivalent to static member variables in C++, since the declaration uses the keyword **static**. In C++, a class variable must be explicitly defined, external to the class declaration. See also instance variable.

ClassWizard file

A file that ClassWizard generates, containing information needed to edit existing classes or add new classes to a project. ClassWizard also uses the ClassWizard file to store information needed to create and edit message maps and dialog data maps, and to create prototype member functions.

ClassWizard files have a .CLW filename extension.

CLI

A library of function calls that support SQL statements and conform to the SQL Access Group Call Level Interface specification. These calls are typically used for dynamic access to data. ODBC is a call level interface.

client

An application or a process that requests a service from some other process, or from an in-process server. See also client/server.

client area

Or client rectangle. The portion of a window where the application displays output such as text or graphics.

client coordinates

An ordered pair (x,y) of numbers, relative to the origin (usually the upper-left corner of a window's client area), that designates a point in the client area. See also window rectangle.

client item

An object that provides an interface between an OLE item and the container application, and that is of a class derived from the MFC class **COleClientItem**. Client items are maintained by the container application and give the container application access to the presentation data and the native data. Client items also provide site(location) information to the server application for in-place activation. See also embedded item, linked item, server item.

client/server

1 The most commonly used model for distributed applications. Client applications request services from a server application. A server can have many clients at the same time, and a client can request data from multiple servers. An application can be both a client and a server. See also client.

2 In network architecture, a model for a local area network where clients initiate communication with the server, which carries out the requests in the form of replies. For example, the clients may be workstations communicating with a file server on which all of their data is stored. See also client.

clip path

A graphics object that an application can select into a device context. A clip path is always created by an application and it is used for clipping to one or more irregular shapes. For example, an application can use the lines and curves that form the outlines of characters in a string of text to define a clip path. See also clipping region.

clipboard

An area of storage, or buffer, where data objects or their references are placed when a user carries out a cut or copy operation.

clipboard format

The data format of a memory object on the clipboard. Applications can use the standard clipboard formats provided by Windows or register their own custom formats. A clipboard format is identified by a unique, unsigned integer value, called the format name.

clipboard owner

Or owner application. The application associated with the information on the clipboard. It is possible for there to be no clipboard owner. See also clipboard viewer.

clipboard viewer

A window that displays the contents of the clipboard. See also clipboard owner.

clipboard-viewer chain

A link between all of the running clipboard-viewer applications, enabling them to all receive the messages that Windows sends to the current clipboard viewer.

clipping

- 1** In Windows, the process of limiting output to a region or path within the client area in a window. For example, word processing and spreadsheet applications clip keyboard input to keep it from appearing in the margins of a page or spreadsheet.
- 2** In Open GL, eliminating the portion of a geometric primitive that is outside the half-space defined by a clipping plane.

clipping precision

A 16-bit value that defines how to clip characters that are partially outside the clipping region.

clipping region

In Windows, the portion of a window's client area where the system permits drawing.

CLSID

A universally unique identifier (UUID) that identifies a type of OLE object. Each type of OLE object (item) has its CLSID in the registry so that it can be loaded and programmed by other applications. For example, a spreadsheet may create worksheet items, chart items, and macrosheet items. Each of these item types has its own CLSID that uniquely identifies it to the system. See also registration entry file.

code page

A character set, which can include numbers, punctuation marks, and other glyphs. Different languages and locales may use different code pages. For example, code page 1252 is used for American English and most European languages. See also locale.

COFF

In 32-bit programming, a format for executable and object files that is portable across platforms. The Microsoft implementation of COFF is derived from the UNIX specification for COFF, but includes additional headers for compatibility with MS-DOS and 16-bit Windows. This Microsoft version is sometimes called the portable executable (PE) file format.

collection class

In object-oriented programming, a class that can hold and process groups of class objects or groups of standard types. A collection class is characterized by its shape (the way the objects are organized and stored) and by the types of its elements. MFC provides three basic collection shapes: lists, arrays, and maps (also known as dictionaries). See also collection object.

collection object

An object in a collection class.

color palette

An array containing the RGB values that identify the colors that can currently be displayed or drawn on the output device. Color palettes are used by devices that are capable of generating many colors but can only display or draw a subset of these at any given time. See also logical color palette.

COM

An open architecture for cross-platform development of client/server applications based on object-oriented technology as agreed upon by Digital Equipment Corporation and Microsoft Corporation. The Component Object Model defines an interface (similar to an abstract base class), **IUnknown**, from which all COM-compatible classes are derived.

combo-box control

In Windows, a child window that consists of a list box combined with either a static control or an edit control. The list-box portion of the control can either be displayed at all times or drop down when the user selects the drop-down arrow next to the control.

COMDAT record

A Common Object File Format (COFF) record that contains initialized common block data and makes packaged functions visible to the linker. See also packaged function.

command file

A text file that contains options and filenames you would otherwise type on the command line or specify using the CL or LINK environment variable. Since the command line is typically limited to 128 characters, a command file allows you to specify a large set of options or a very long file list to the compiler, linker, or resource compiler, for instance.

command identifier

Or command ID. In MFC, an identifier that associates a command message with the user-interface object (such as a menu item, toolbar button, or accelerator key) that generated the command. Typically, command IDs are named for the functionality of the user-interface object they are assigned to. For example, a Clear All item in the Edit menu might be assigned an ID such as ID_EDIT_CLEAR_ALL.

command line

A string of text typed at the command prompt, or executed from a command file, that specifies a task or tasks for the operating system or an application to perform.

command message

1 In Windows, a notification message from a user-interface object, such as a menu, toolbar button, or accelerator key. The framework processes command messages differently from other messages and such messages can be handled by a wider variety of object—documents, document templates, and the application object itself, in addition to windows and views.

2 In Media Control Interface (MCI), a symbolic constant that represents a unique command for an MCI device. Command messages have associated data structures that provide information a device requires to carry out a request.

comment delimiters

Characters used to denote text in a program that is not source code, thus telling the compiler to ignore it. C++ allows the traditional comment delimiters:

```
/* this is a comment */
```

as well as a single-line comment delimiter:

```
// everything else on this line is a comment
```


commit size

The amount of a resource that is allocated (or committed) for a particular use. For example, in the header of a COFF file, the Windows NT–specific field Heap Commit Size specifies the size of the local heap that the linker and loader should allocate for that file. See also reserve size.

common data record

A Common Object File Format (COFF) record that contains initialized common block data and makes packaged functions visible to the linker. See also packaged function.

common dialog box

A dialog box predefined in Windows that supports standard operations, such as the Open command on the File menu. An application displays a common dialog box by calling a single function rather than by supplying a dialog box procedure and using a resource file containing a dialog box template.

Common Gateway Interface

A mechanism that allows a Web server to run a program or script on the server and send the output to a Web browser. See also Internet Server Application Programming Interface (ISAPI).

Common Object File Format

In 32-bit programming, a format for executable and object files that is portable across platforms. The Microsoft implementation of COFF is derived from the UNIX specification for COFF, but includes additional headers for compatibility with MS-DOS and 16-bit Windows. This Microsoft version is sometimes called the portable executable (PE) file format.

compact executable file

An executable binary (program) file whose code is limited to a single 64-kilobyte segment. Compact executable files usually have a .COM filename extension and are often used for utility programs and short routines. See also executable file.

compilation

The translation of source code into object code.

compilation unit

The smallest unit of code that can be independently compiled, usually a source code file. See also translation unit.

compile time

The point at which a program is being compiled, or the amount of time required to perform a compilation of a program.

compile-time error

A syntactic or semantic error that prevents a program from being compiled.

compiled resource file

Or binary resource file. A binary file that contains a Windows-based application's resource data and is created by the resource compiler from the resource-definition (.RC) file. Compiled resource files usually have a .RES filename extension. See also Macintosh binary resource file, resource compiler.

compiler

A program that translates source code, such as C++ or Pascal, into directly executable machine code.

compiler/linker driver

Or CL. A driver program that controls the Microsoft C and C++ compilers and linker. The compilers produce Common Object File Format (COFF) object files. The linker produces executable (.EXE) files, dynamic-link libraries (DLLs), or static-link libraries.

complete object

An instance of a derived class from which no other classes are derived. A complete object is an object that is not a subobject representing a base class.

complex type

See aggregate type.

Component Object Model

An open architecture for cross-platform development of client/server applications based on object-oriented technology as agreed upon by Digital Equipment Corporation and Microsoft Corporation. The Component Object Model defines an interface (similar to an abstract base class), **IUnknown**, from which all COM-compatible classes are derived.

compositing

The process of superimposing one image on another to create a single image.

compound document

Or container document. A document within a container application that contains data of different formats, such as sound clips, spreadsheets, text, and bitmaps. Each piece of integrated data (or compound-document object) can exist within the compound document as a linked item or an embedded item.

compound file

The OLE implementation of the structured-storage model, which specifies how data is saved to and retrieved from storage. Conceptually, a compound file is a number of individual files (or stream objects) multiplexed into one physical file (or storage object) that still allows access to each individual file.

Compressed SLIP

SLIP with data compression for a more efficient connection. See also Serial Line Internet Protocol (SLIP).

conditional expression

Or Boolean expression, logical expression. An expression that yields a Boolean value (true or false). Such expressions can involve comparisons, using relational operators such as the less-than (<) and greater-than (>) operators, and logical combination of Boolean expressions, using Boolean operators such as bitwise AND (&) and logical OR (||).

connection string

Or connect string. In ODBC, a string expression used to open an external database.

console

An interface that provides input and output to character-mode applications. This processor-independent mechanism makes it easy to port existing character-mode applications or to create new character-mode tools and applications.

console application

- 1** A character-mode application that uses a console window for its input and output. If necessary, the operating system will create a new console window, which exists until the application terminates.
- 2** More generally, a program that runs from the operating system's command line, in character-mode, rather than from a graphical user interface.

constant

An object or variable that is not modifiable. In C++, the keyword **const** can be used to define constant values. See also manifest constant.

constant expression

An expression that is evaluated at compile time instead of run time. The constant expression can be used in any place that a constant can be used, but it must evaluate to a constant that is in the range of representable values for that type.

constructor

In C++, a special initialization function that is called automatically whenever an instance of a class is declared. This function prevents errors that result from the use of uninitialized objects. The constructor must have the same name as the class itself and must not return a value. See also copy constructor, default constructor, destructor.

container application

Or OLE container. An application that can incorporate embedded or linked items into its own documents. The documents managed by a container application are able to store and display OLE Visual Editing items as well as data created by the application itself. A container application allows users to insert new items or edit existing items. See also server application.

context identifier (ID)

Or context reference. A unique number or string that corresponds to a particular object in the application—for example, to a menu command, form, control, or screen region. Context IDs are used to create links between the application and the corresponding Help topics.

context number

The number used to identify a Windows Help topic. If context numbers are not explicitly assigned to topics, the Help compiler generates default values by converting topic strings into context numbers. The [MAP] section of a Help project (.HPJ) file associates a context string and a context number. See also context string.

context string

A unique character string formatted as hidden text in a rich-text format (.RTF) file. Context strings link hot spots to target topics. The [MAP] section of a Help project (.HPJ) file associates a context string and a context number. See also context number.

control bar

A window that can contain buttons, edit boxes, check boxes, or other kinds of Windows controls. A control bar is usually aligned with the top or bottom of a frame window and provides quick, one-step command actions. Control bars include toolbars, status bars, and dialog bars.

control identifier (ID)

A 16-bit value that an application uses to uniquely identify a child control. This ID is used in notification messages to the parent window when events, such as input from the user, occur in the control.

conversion function

1 In C++, a special member function that makes an explicit conversion from a given class type to another data type by using the **operator** *type-name*() syntax. Conversion functions are often called cast operators because they are the functions called when a cast operator is used.

2 More generally, any function that converts one data type or format to another data type or format.

coordinate space

A planar space based on the Cartesian coordinate system.

coordinated universal time

A global time standard equivalent to Greenwich mean time (GMT).

copy constructor

In C++, a constructor with one parameter, whose type is a reference to another instance of the class. If a class is declared without a copy constructor, the compiler will generate one automatically. The copy constructor is used when an instance is created from another instance (for example, an assignment from one instance of the class to another).

CRC

An error-detecting method that uses a polynomial code. The method is sometimes referred to as the polynomial code.

critical section

A segment of code which is not reentrant; that is, it does not support concurrent access by multiple threads. Often, a critical section is used to protect shared resources

cross compilation

A compilation of source code that takes place on one hardware platform but generates object code for another. For example, object code for the Power Macintosh can be compiled on an Intel-based Windows platform. See also compilation, object code.

CSLIP

SLIP with data compression for a more efficient connection. See also Serial Line Internet Protocol (SLIP).

cursor resource file

A file that contains an image that defines the shape of a cursor on the screen. Cursor resource files usually have a .CUR filename extension.

custom control

A special-format dynamic-link library (DLL) or object file that adds features and functionality to a Windows-based application user interface. A custom control can be a variation on an existing Windows dialog-box control (for example, a text box suitable for use with a pen and digitizing tablet) or an entirely new category of control. See also ActiveX control.

custom resource

Or application-defined resource. A resource that a developer creates and adds to an application's executable file that contains data required by the application. See also standard resource.

cyclic redundancy check

An error-detecting method that uses a polynomial code. The method is sometimes referred to as the polynomial code.

DAG

In programming, an abstract data type often used to represent arbitrary relationships between objects. The graph consists of nodes (data objects) connected by paths that have direction; there are no cycles in the graph. In object-oriented program design, DAGs are useful for depicting inheritance relationships among classes.

DAO

A high-level set of objects that insulates developers from the physical details of reading and writing records. In a database application, for example, these objects include databases, table definitions, query definitions, fields, indexes, etc.

DAO record field exchange

The mechanism by which the MFC DAO classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), bulk record field exchange (Bulk RFX), and dialog data exchange (DDX).

Data Access Objects

A high-level set of objects that insulates developers from the physical details of reading and writing records. In a database application, for example, these objects include databases, table definitions, query definitions, fields, indexes, etc.

data binding

A notification mechanism that links control properties through the container to a data source, such as a database field.

data declaration

A statement within a program that specifies the characteristics of a variable. Most programming languages allow (or require) specification of a variable's name and data type and possibly its initial value as well. Array declarations usually require a size specification in addition to the name and type, and record declarations must specify the elements of the record. See also data type, type declaration.

data definition language

Or database design language, data design language. A language, usually a part of a database management system, that defines all attributes and properties of a database, especially record layouts, field definitions, key fields (and, sometimes, keying methodology), file locations, and storage strategy.

data field

- 1 A well-defined portion of a data record, such as a column in a database table.
- 2 The physical representation of a unit of data.

data file

A file consisting of data—text, numbers, or graphics, for example. Such a file is distinct from a program file of executable instructions.

data fork

The portion of an Apple Macintosh file that contains user-supplied information. See also resource fork.

data handle

In Microsoft Windows, a 32-bit value used to provide access to data in a dynamic data exchange (DDE) object.

data map

In MFC, a mechanism that automates the process of gathering values from a dialog box by providing functions to initialize the controls in the dialog box with the proper values, retrieve the data, and validate the data.

data member

A data object defined as part of a class. See also member function.

data segment

The portion of memory or auxiliary storage that contains the data needed by an application or a portion of an application. Usually, the data segment is readable and writable.

data source

In ODBC, a specific set of data, the information required to access that data, and the location of the data source, which can be described using a data source name. From a program's point of view, the data source includes the data, the DBMS, the network (if any), and ODBC.

data source name

The name of a data source that applications use to request a connection to the data source. For example, a data source name can be registered with ODBC through the ODBC Administrator program.

data structure

An organizational scheme, such as a record or an array, applied to data so that it can be interpreted and so that specific operations can be performed on that data.

data symbol

The name that identifies the memory location of a global or static data object. The concept of data symbol includes all data objects except local (stack-allocated) or dynamically allocated data.

data type

In programming, a definition of a set of data that specifies the possible range of values of the set, the operations that can be performed on the values, and the way in which the values are stored in memory. See also aggregate type, scalar type.

data-access application

An application used to access data, typically in a database. See also database application.

database application

An application that manages files consisting of a number of records (or tables), each of which is constructed of fields (columns) of a particular type, together with a collection of operations that facilitate searching, sorting, recombination, and similar activities. See also data-access application, relational database.

database form

A structured window, box, or other self-contained presentation element built into a database application that allows the user to perform a variety of data-access tasks, including data entry, read-only examination of data, and data updates. The form acts as a visual filter for the underlying data it is presenting, generally offering the advantages of better data organization and greater ease of viewing.

database management system

A layer of software between the physical database and the user. The DBMS manages all requests for database action (for example, queries or updates) from the user. Thus, the user is spared the necessity of keeping track of the physical details of file locations and formats, indexing schemes, and so on. In addition, a DBMS may permit centralized control of security and data-integrity requirements.

database schema

A description of the current structure of tables and views in a database. The schema describes the columns in each table and the data type of each column.

datagram socket

A connectionless socket that provides a bidirectional flow of data. Datagram data may arrive out of order and possibly duplicated, but record boundaries in the data are preserved, as long as the records are smaller than the receiver's internal size limit. An example of a datagram socket is an application that keeps system clocks on the network synchronized. See also stream socket, transport protocol.

DBCS

A character set that can be used to represent Far Eastern languages that use ideographic characters. Like a multibyte character set (MBCS), a DBCS contains both single- and double-byte characters. DBCS characters are addressed using two bytes. The DBCS single-byte characters conform to the 8-bit national standards for each country and correspond closely to the ASCII character set. See also lead byte, trail byte, Unicode.

DBMS

A layer of software between the physical database and the user. The DBMS manages all requests for database action (for example, queries or updates) from the user. Thus, the user is spared the necessity of keeping track of the physical details of file locations and formats, indexing schemes, and so on. In addition, a DBMS may permit centralized control of security and data-integrity requirements.

DC

A data structure defining the graphic objects, their associated attributes, and the graphic modes affecting output on a device. See also graphic object, metafile.

DDE

A form of interprocess communications that uses shared memory to exchange data between applications. DDE can be used for one-time data transfers and for ongoing exchanges by applications that send updates to one another as new data becomes available. See also client/server, OLE Automation.

DDL

Or database design language, data design language. A language, usually a part of a database management system, that defines all attributes and properties of a database, especially record layouts, field definitions, key fields (and, sometimes, keying methodology), file locations, and storage strategy.

DDV

In MFC, a method for checking data as it is transferred from the controls in a dialog box. DDV is an easy way to validate data entry in a dialog box. See also dialog data exchange (DDX).

DDX

In MFC, a method for transferring data between the controls of a dialog box and their associated variables. DDX is an easy way to initialize the controls in a dialog box and to gather data input by the user. See also dialog data validation (DDV).

dead key

On some non-English keyboards, a key that is used with another key to create an accented character. A dead key, when pressed, produces no visible character but indicates that the diacritic it represents is to be combined with the character produced by the next letter key pressed.

deadlock

A state in which every process in a set of processes is waiting for an event or resource that only another process in the set can provide. For example, in data communications, a deadlock can occur when both the sending and receiving sockets are waiting on each other, or for a common resource.

debug memory allocator

A memory allocation function that allows you to validate and track the memory allocated for your objects on the global heap. For example, the Debug version of the Microsoft Foundation Class Library has memory allocation functions.

debug monitor

Or remote monitor. A small program that resides on the target machine and controls the execution of the remote program and communication with the Microsoft Developer Studio debugger. See also remote debugging.

debug terminal

A dumb terminal or computer other than the developer's primary machine, to which debugging output is sent.

debug version

- 1** A version of a program built with symbolic debugging information. See also release version.
- 2** Or debug library. A library version (for example, of the Microsoft Foundation Class Library) that includes diagnostic aids and performs various integrity checks to aid in debugging a program. See also release version.

debugger

A program designed to help find errors in another program by allowing the programmer to step through the program, examine data, and check conditions. There are two basic types of debuggers. Machine-level debuggers display the actual machine instructions (disassembled into assembly language) and allow the programmer to look at registers and memory locations. Source-level debuggers let programmers look at the original source code, examine variables and data structures by name, and so on.

debugging event

An incident in the process being debugged that causes the kernel to notify the debugger. Debugging events include creating a process, creating a thread, loading a dynamic-link library (DLL), unloading a DLL, sending an output string, and generating an exception.

debugging information

Symbolic information used by a debugger, especially information in the Microsoft Symbolic Debugging Information format that is used by the Microsoft CodeView debugger. Also, any data (information) generated by the debugging process.

declaration

A statement that binds an identifier to the information that relates to it. For example, to declare a constant means to bind the name of the constant with its value. To declare a variable means to bind the variable's name with a location in memory and with the information about the variable's data type. Declaration usually occurs within the source code of a program; the actual binding can take place at compile time or run time. Declaration can be performed explicitly (by specifying the identifier and relevant information in a declare statement) or implicitly (by using the undeclared identifier in a statement), depending on the language being used.

declaration statement

A statement used to declare the names and data types of constants, variables, user-defined data types, and procedures. In C++, the declaration statement can also be an executable statement. For example:

```
int i = fCount( );
```

declares an integer variable `i` and assigns it the value returned from the call to `fCount()`.

declarator

The part of a declaration that names an object, type, or function. In C/C++, declarators appear in a declaration as one or more names separated by commas; each name can have an associated initializer.

decorated name

Or type-safe name, mangled name. In C++, a compiler-generated string that contains an undecorated (literal) name followed by a string of characters that the compiler and linker use to retain type information. Name decoration is necessary to resolve ambiguities that arise in C++ from having more than one function, for example, with the same name.

decrement (--) operator

An operator that specifies that its integral or floating-point operand be decreased by the integer value 1. A decremented pointer points to the previous object. In the prefix form (`--i`), the decrement takes place before the value is used in expression evaluation; in the postfix form (`i--`), the decrement takes place after the value is used in expression evaluation. See also increment (`++`) operator.

default argument

In a C++ function definition, an argument with an assigned value that specifies the value an argument should assume if none is supplied in the function call.

default constructor

In C++, a constructor that either accepts no arguments or for which all arguments have a default value. The default constructor can be defined by the user or generated by the compiler. See also constructor, copy constructor.

default message processing

Message processing carried out by a default window procedure on any window messages for which there is not an explicit procedure defined in the application. In applications based on the Microsoft Foundation Class Library, **DefWindowProc** is the default window procedure.

default value

A value that the system or software assumes, unless the user makes an explicit choice.

default window procedure

A system-defined function that defines certain fundamental behavior shared by all windows. A default window procedure provides default processing for nonclient-area messages, system commands, system keystrokes, and other messages that the application-defined window procedure does not specifically handle. See also window procedure.

definition

A construct that initializes and allocates storage for a variable, function, or class. For a function, the definition specifies the name, formal parameters, body, and return type; for a class, the definition specifies its data members and functions. See also declaration.

delayed rendering

Formatting data when it is requested, rather than when it first becomes available. For example, when placing a Clipboard format on the Clipboard, a window can delay transferring the data to that format until the data is needed. This is useful if the application supports several Clipboard formats, some or all of which are time-consuming to render.

delimiter

A special character that sets off, or separates, individual items in a program or in a set of data. Programming languages typically delimit such variable-length elements as comments, strings, and program blocks. Databases use two forms of delimiters: field delimiters and record delimiters. Characters used as delimiters include commas, semicolons, tabs, carriage returns, and colons.

derived class

In object-oriented programming, a class that is created from another class, called the base class. A derived class inherits all the features of its base class. See also inheritance, polymorphism.

descendant

- 1 In object-oriented programming, a class that is derived from another class. See also base class, derived class, inheritance.
- 2 Generally, a process or task that is called by another process or task and whose characteristics and behavior are determined, at least in part, by the originating process or task. See also child process, child window.

descendant window

A child window that is derived from a parent window. A child window has only one parent window, but a parent can have any number of child windows. Each child window, in turn, can have child windows. In this chain of windows, each child window is called a descendant window of the original parent window.

description block

In a makefile, a dependency line that specifies a block of commands to run if the listed targets and their dependents are out of date.

deserialization

Re-creating an object by reading its state from persistent storage (usually a file) and reconstructing it in memory. See also serialization.

Desktop Management Interface

A protocol-independent, multiplatform interface for workstation and network components.

desktop window

A system-defined window that paints the background of the screen and serves as the base for all windows displayed by all Windows-based applications.

destructor

In C++, a member function that is automatically called when a class object is destroyed (either by being explicitly deallocated or by going out of scope). Its purpose is to perform any cleanup work necessary before an object is destroyed. The destructor's name is the class name with a tilde (~) as a prefix. See also constructor.

development environment

In the Microsoft Developer Studio, an integrated set of Windows-based tools for completing, testing, and refining an application. Microsoft Developer Studio includes a text editor, resource editors, project build facilities, an optimizing compiler, an incremental linker, a source code browse window, and an integrated debugger.

device context

A data structure defining the graphic objects, their associated attributes, and the graphic modes affecting output on a device. See also graphic object, metafile.

device driver

A low-level software component that permits device-independent software applications to communicate with a device such as a mouse, keyboard, monitor, or printer.

device-independent

A characteristic of software and files that generate the same output regardless of the hardware involved. A device-independent program could, for example, issue the same command to draw a rectangle regardless of whether the output device was a printer, a plotter, or a screen display.

device-independent bitmap file

A file containing an array of bits combined with several structures that specify the width and height of the bitmapped image (in pixels), the color format of the device where the image was created, and the resolution of the device used to create that image. The DIB file format ensures that bitmap graphics created in one application can be loaded and displayed in another application exactly the way they appear in the originating application. See also bitmap, bitmap file.

device-mode setting

A user-selected setting that contains information used by the relevant device driver. For example, device-mode settings for a laser printer setup could specify whether printing is to be in landscape or portrait mode, the size of paper used, and the quality of printing.

DFX

The mechanism by which the MFC DAO classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also record field exchange (RFX), bulk record field exchange (Bulk RFX), and dialog data exchange (DDX).

diagnostic services

Services that facilitate program debugging. Diagnostic services include functions and macros to resolve assertions, errors, and exceptions; trace memory allocations; and resolve leaks; and report debug messages to the user—all during run time. In Microsoft Visual C++, most of these services require the debug version of the C run-time library.

dial on demand

In ISDN, a dial-up router function that activates a remote link only when data needs to be sent.

dialog bar

A control bar that contains standard Windows controls. A dialog bar has dialog-box characteristics in that it contains controls and supports tabbing between them, and it uses a dialog template to represent the bar. Dialog bars can be aligned to the top, bottom, left, or right side of a frame window. See also status bar.

dialog box

In Windows, a child window used to retrieve user input. A dialog box usually contains one or more controls, such as buttons, list boxes, combo boxes, and edit boxes, with which the user enters text, chooses options, or directs the action of the command.

dialog data exchange

In MFC, a method for transferring data between the controls of a dialog box and their associated variables. DDX is an easy way to initialize the controls in a dialog box and to gather data input by the user. See also dialog data validation (DDV).

dialog data validation

In MFC, a method for checking data as it is transferred from the controls in a dialog box. DDV is an easy way to validate data entry in a dialog box. See also dialog data exchange (DDX).

dialog editor

A resource editor that allows you to place and arrange controls in a dialog-box template and to test the dialog box. The editor displays the dialog box exactly as the user will see it. While using the dialog editor, you can define message handlers and manage data gathering and validation with the ClassWizard.

dialog file

A file that contains dialog-box source code. Note that a dialog file is not required for a Visual C++ project because Visual C++ keeps this code in the resource-definition file.

dialog template

A template used by Windows to create a dialog window and display it. The template specifies the characteristics of the dialog box, including its overall size, initial location, and style, and the types and positions of its controls. A dialog template is usually stored as a resource, but templates can also be stored directly in memory. See also [dialog file](#), [dialog editor](#), [resource-definition file](#).

dialog unit

A unit of horizontal or vertical distance within a dialog box. A horizontal DLU is the average width of the current dialog-box font divided by 4. A vertical DLU is the average height of the current dialog-box font divided by 8.

digital signature

An electronic identifier used for security. It verifies that the document originated from the individual whose signature is attached to it and that it has not been altered since it was signed. Usually accomplished using some form of encryption.

dimension

The zero-based number or numbers inside brackets in an array declaration that define the array's size. For example, **char p[10]** declares a one-dimensional character array that has ten elements. To declare arrays of two or more dimensions, place each dimension in its own set of brackets, as in **char x[10][20]**.

dimmed

Or disabled, grayed, unavailable. The state and visual appearance of controls or menu items whose functionality is not presently available to a user.

direct mode

A file-access mode that saves changes to the document as they are made. See also transacted mode.

directed acyclic graph

In programming, an abstract data type often used to represent arbitrary relationships between objects. The graph consists of nodes (data objects) connected by paths that have direction; there are no cycles in the graph. In object-oriented program design, DAGs are useful for depicting inheritance relationships among classes.

dirty

Indicates that a file, object, or data item has been changed since the last time it was saved. Usually describes a flag or bit that is set to 1 when data changes and back to 0 when the data is saved to disk.

disjoint figure

A figure that consists of a set of points that are not logically sequential or physically adjacent to each other.

disk drive

A physical device that reads from or writes to disks. Disk drives are referenced by letters, typically A: for the first floppy-disk drive, B: for the second floppy-disk drive, C: for the first fixed-disk drive, D: for the second fixed-disk drive, and so on. Each drive (whether physical or logical) on an operating system is assigned a unique letter to distinguish it from other drives.

dispatch identifier (ID)

A 32-bit attribute value for identifying methods and properties in OLE Automation. All of the accessor functions for a single property have the same dispatch ID. The low-order 16 bits of the dispatch ID contain the distance from the top of the dispatch map; the high-order 16 bits contain the distance from the most derived class.

dispatch interface

In OLE Automation, the external programming interface of some grouping of functionality exposed by the automation server. For example, a dispatch interface might expose an application's mouse clicking and text data entry functions. See also type library (.TLB) file.

dispatch map

In MFC, a set of macros that expands into the declarations and calls needed to expose methods and properties for OLE Automation. The dispatch map designates the internal and external names of object functions and properties, as well as the data types of the function arguments and properties.

display device context

A device context, created by Windows, that an application uses to paint and draw on a video display. Windows prepares a display device context for each window, or the screen, setting the drawing objects, colors, and modes for the device.

Distributed Management Environment

A strategy developed to manage distributed heterogeneous networks that conform to DCE.

dithering

A technique for increasing the perceived range of colors in an image at the cost of spatial resolution. Adjacent pixels are assigned differing color values; when viewed from a distance, these colors seem to blend into a single intermediate color. The technique is similar to the half-toning used in black-and-white publications to achieve shades of gray.

DLU

A unit of horizontal or vertical distance within a dialog box. A horizontal DLU is the average width of the current dialog-box font divided by 4. A vertical DLU is the average height of the current dialog-box font divided by 8.

DME

A strategy developed to manage distributed heterogeneous networks that conform to DCE.

DMI

A protocol-independent, multiplatform interface for workstation and network components.

dockable toolbar

A toolbar that can be attached, or docked, to any side of its parent window, or floated in its own mini-frame window. See also docked toolbar, floating toolbar.

docked toolbar

A toolbar that is attached, or docked, to any side of its parent window. See also dockable toolbar, floating toolbar.

document

1 (noun) Any self-contained piece of work created with an application program and, if saved on disk, given a unique filename by which it can be retrieved.

2 (verb) To explain or annotate something, such as comments within a program or a description of a problem in a bug report.

document item

An object of a class derived from the **CDocItem** class, encapsulating some component of a document's data. Document items are used to represent OLE items in both client and server documents. See also client item, server item.

document object

An object that defines, stores, and manages an application's data. When the user opens an existing or new document, the application framework creates a document object to manage the data stored in the document.

document template

In MFC, a template used for the creation of documents, views, and frame windows. A single application object manages one or more document templates, each of which is used to create and manage one or more documents (depending on whether the application is SDI or MDI).

Applications that support more than one type of document, such as spreadsheets as well as text, have multiple document template objects.

document window

In windowing environments, an on-screen window (enclosed work area) in which the user can create, view, or work on a document.

document/view architecture

A design methodology that focuses on what the user sees and needs rather than on the application or what the application requires. This design is implemented by a set of classes that manage, store and present application-specific data. These classes can manipulate disk-based data files (document objects), display a document's data (view objects), and automatically use a particular type of window (window objects).

domain name

- 1** In general networking, a logical grouping of machines for network administration purposes.
- 2** In TCP/IP, the unique name that identifies an Internet or network site. Domain names always have two or more parts, separated by dots (for example, **microsoft.com** or **www.microsoft.com**). The part on the left is the most specific, and the part on the right is the most general. A given machine may have more than one domain name, but a given domain name points to only one machine.

dots per inch

A measure of screen and printer resolution that is expressed as the number of dots per horizontal or vertical inch that a device can print or display .

double-byte character set

A character set that can be used to represent Far Eastern languages that use ideographic characters. Like a multibyte character set (MBCS), a DBCS contains both single- and double-byte characters. DBCS characters are addressed using two bytes. The DBCS single-byte characters conform to the 8-bit national standards for each country and correspond closely to the ASCII character set. See also lead byte, trail byte, Unicode.

doubleword

Or DWORD. A unit of data consisting of 4 contiguous bytes that are processed as a single unit by a computer's microprocessor.

DPI

A measure of screen and printer resolution that is expressed as the number of dots per horizontal or vertical inch that a device can print or display .

drag-and-drop

A technique for moving or copying data between applications, between windows within an application, or within a single window in an application. The user selects the data to be transferred and drags the data to the desired destination.

driver

- 1** A hardware device that controls or regulates another device. A line driver, for example, boosts signals transmitted over a communications line, and a bus driver amplifies and regulates signals transmitted over a bus (data pathway).
- 2** A program that controls a device such as a printer or a mouse. See also device driver.

drop source

In a drag-and-drop operation, the window from which the user selects data for transfer. See also drop target.

drop target

In a drag-and-drop operation, the destination window where the user drops the data being transferred. See also drop source.

drop-down combo box

A combo box that contains a drop-down list and a selection field that the user can edit.

drop-down list

A list in a combo box that displays the current setting, but can be opened to display a list of choices. The user can select an item from the list to update the current setting.

drop-down menu

A menu that is displayed when the user selects a particular entry from a menu bar. See also pop-up menu.

DSN

The name of a data source that applications use to request a connection to the data source. For example, a data source name can be registered with ODBC through the ODBC Administrator program.

dual interface

An interface that supports both IDispatch and VTBL binding.

DUMPBIN

The Microsoft COFF Binary File Dumper (DUMPBIN.EXE). DUMPBIN displays information about 32-bit Common Object File Format (COFF) binary files. DUMPBIN can be used to examine COFF object files, standard libraries of COFF objects, executable files, and dynamic-link libraries (DLLs). DUMPBIN is a 32-bit tool that runs only from a command prompt.

dynamic creation

The process of creating an object of a specific class at run time. Do not confuse this dynamic creation of an object with the creation of a dynamic object, using the C++ **new** operator. Dynamic creation is not supported directly by the C++ language. Objects derived from the MFC class **CObject** can have this functionality.

dynamic data exchange

A form of interprocess communications that uses shared memory to exchange data between applications. DDE can be used for one-time data transfers and for ongoing exchanges by applications that send updates to one another as new data becomes available. See also client/server, OLE Automation.

dynamic link

A program link, to a library or to an object, that is established when the program is loaded into memory (load-time dynamic linking) or while it is running (run-time dynamic linking). See also dynamic-link library (.DLL) file, static link.

dynamic priority

A thread priority value used by the scheduler in making scheduling decisions. The value for each thread can never be lower than the thread's base priority, but it can be raised and then lowered to enhance responsiveness to input or other significant events.

dynamic splitter window

A split-window style in which additional panes are created and destroyed as the user splits and unsplit views. Microsoft Excel and Microsoft Word are examples of applications that offer the dynamic splitter style. See also static splitter window.

dynamic-link library file

A file that contains one or more functions that are compiled, linked, and stored separately from the processes that use them. In Win32, the operating system maps the dynamic-link libraries (DLLs) into the address space of a process when the process is starting up or while it is running. The process then executes functions in the DLL. Dynamic-link library files usually have a .DLL filename extension.

dynaset

A recordset (or set of records) with dynamic properties that is the result of a query on a database document. A dynaset can be used to add, change, and delete records from the underlying database table or tables. See also snapshot.

edit buffer

In MFC database classes, a buffer that contains the current record during an update. Update operations may use this buffer to manage changes to the data source.

edit control

Or edit box, text box. A rectangular control window that a user can use to type and edit text. See also static control.

EDITBIN

Or COFF Binary File Editor. The Microsoft 32-Bit binary file editor for modifying 32-bit Common Object File Format (COFF) binary files. Use EDITBIN to modify object files, executable files, and dynamic-link libraries (DLLs). You can also use EDITBIN to convert the format of an Object Model Format (OMF) input file to COFF.

embedded item

Or embedded object. A type of compound-document item in which all the information needed to manage the item is stored in the container document, but which is created and edited by a server application. Embedded items can be edited or activated in-place. See also linked item.

encapsulation

In object-oriented programming, the process of hiding the internal workings of a class to support or enforce abstraction. A class's interface, which is public, describes what a class can do, while the implementation, which is private or protected, describes how it works.

encryption

The transformation of data into a form unintelligible to anyone without a secret decryption key and algorithm. Its purpose is to ensure privacy by keeping the information hidden from anyone for whom it is not intended.

end of file

A value returned by an I/O routine when the end of a file (or, in some cases, an error) is encountered. The iostream library function **eof** returns TRUE on the end-of-file condition. When a file is opened in text mode, a logical end of file occurs whenever a CTRL+Z character is encountered.

entry point

A starting address for a function, executable file, or dynamic-link library.

environment variable

A symbolic variable that represents an element of the user's operating system environment, such as a path, a directory name, or a configuration string. For example, the environment variable PATH represents the directories to search for executable files.

EOF

A value returned by an I/O routine when the end of a file (or, in some cases, an error) is encountered. The iostream library function **eof** returns TRUE on the end-of-file condition. When a file is opened in text mode, a logical end of file occurs whenever a CTRL+Z character is encountered.

epilog code

See prolog/epilog code sequence.

error message

A message from the system or a program advising the user of a problem that requires human intervention in order to be solved.

escape sequence

In C/C++, a character combination consisting of a backslash (\) followed by a letter or by a combination of digits. An escape sequence is regarded as a single character and is therefore valid as a character constant. Escape sequences are typically used to provide literal representations of nonprinting characters, such as the newline character (\n) and characters that have special meanings in the C/C++ language, such as the double quotation mark (\").

event

- 1** In OLE, a notification message sent from one object to another (e.g. from a control to its container) in response to a state change or a user action.
- 2** More generally, any action or occurrence, often generated by the user, to which a program might respond. Typical events include keystrokes, mouse movements, and button clicks.

event object

A synchronization object that allows one thread to notify another that an event has occurred. Event objects are useful when a thread needs to know when to perform its task. For example, a thread that copies data to a data archive would need to be notified when new data is available. By using an event object to notify the copy thread when new data is available, the thread can perform its task as soon as possible.

exception

An abnormal condition or error that occurs during the execution of a program and that requires the execution of software outside the normal flow of control. Examples of exceptions are running out of memory, resource allocation errors, and failure to find files. See also C++ exception handling, structured exception handling (SEH).

exception handler

A block of code that reacts to a specific type of exception. If the exception is for an error from which the program can recover, the program can resume executing after the exception handler has executed. In this case, execution will resume where the exception was handled, not at the place where it was generated.

executable file

A program file created from one or more source code files translated into machine code and linked together. The MS-DOS, Windows, and Windows NT operating systems use the .EXE filename extension to indicate that the file is a runnable program.

execution character set

The character set on the machine where the program executes. For Microsoft C and C++, the execution set is the standard ASCII character set. See also source character set.

explicit initializer

An initial value that is explicitly stated when a program variable is declared. In C++, a single initializer can be supplied with the simple-assignment (=) operator, as follows:

```
int nCount = 0;
```

or an initializer list can be supplied, enclosed in parentheses:

```
CString strFileName(FILE.DAT);  
CRect *pRect = new CRect(10, 15, 24, 97);
```


exported function

One called by other executing entities, such as DLLs or executable files. Typically, a DLL will need to export functions to allow its clients to control it. Functions can be exported by using either a module-definition (.DEF) file or the `__declspec (dllexport)` storage-class modifier, a Microsoft-specific extension to the C language. This modifier ensures that other applications and dynamic-link libraries will be able to obtain, dynamically at runtime, the address of the variable or function to which it applies. See also imported function.

exports file

A file that contains information about exported functions and data items. The Microsoft 32-Bit Library Manager tool (LIB.EXE) generates the exports file from the module-definition (.DEF) file. The linker uses the exports file to build the dynamic-link library (.DLL) file. Exports files have a .EXP filename extension.

expression

A sequence of tokens that can be evaluated.

expression statement

An expression followed by a semicolon (;). All expressions in an expression statement are evaluated and all side effects are completed before the next statement is executed. The most common expression statements are assignments and function calls.

external linkage

Specifies the way the names of objects and functions are shared between translation units. With external linkage, names can refer to program elements in any translation unit in the program—the program element is shared among the translation units and the same name in another translation unit is guaranteed to refer to the same object or class. Names with external linkage are sometimes termed global. The keyword **extern** before a name declaration ensures external linkage. See also internal linkage.

external name

- 1 In OLE Automation, an identifier that a class exposes to other applications. Automation clients use the external name to request an object of this class from an automation server.
- 2 In C/C++, an identifier declared with global scope or declared using the **extern** storage class.

extraction (>>) operator

Or get-from operator, input operator. In C++, the right-shift operator, overloaded (in the iostream library) to accept input. The left operand must be the predefined input stream **cin** and the right operand must be a variable. See also insertion (<<) operator.

F1 Help

Context-sensitive Windows Help that the user obtains by pressing the F1 key. F1 Help opens Help on a topic associated with the currently selected item in the application. MFC supplies F1 Help for windows, dialog boxes, message boxes, menus, and toolbar buttons.

FAQ

Documents that list and answer the most common questions on a particular subject.

far pointer

Or long pointer. In 16-bit programming, a 32-bit pointer, which is a pointer that can point anywhere in memory because it specifies both the segment and the offset for a memory location. In 32-bit programming, all pointers are 32 bits wide and there is no need to distinguish between near and far pointers.

FAT

A data member of a recordset object that corresponds to a particular column (or field) in the query that the recordset represents. Collectively, these field data members make up a buffer that holds the fields of the current record. When the user scrolls to a new record, the database framework replaces the previous record's values with the values of the new current record.

fatal error

Or unrecoverable error. An error that causes the system or a program to fail abruptly with no hope of recovery. An example of a fatal error is an uncaught exception that cannot be handled.

favorite

A reference in Internet Explorer to a page to which the user may want to return. Corresponds to bookmark in other browsers.

field data member

A data member of a recordset object that corresponds to a particular column (or field) in the query that the recordset represents. Collectively, these field data members make up a buffer that holds the fields of the current record. When the user scrolls to a new record, the database framework replaces the previous record's values with the values of the new current record.

file allocation table file system

The file system that MS-DOS uses to store information on disks. The file allocation table, which the operating system creates when it formats a disk, holds information about the location of each file as it is stored. See also New Technology file system (NTFS), high-performance file system (HPFS).

file buffer

A reserved portion of memory used to temporarily store data, pending an instruction to complete its transfer to or from a file.

file exception

An exception (abnormal condition or error) that occurs during the course of opening, reading from, or writing to a file. See also C++ exception handling.

file handle

A unique identifier that Windows assigns to a file when the file is opened or created. A file handle is valid until the file is closed.

file input/output (I/O)

The mechanism for making data persistent between program work sessions by creating files, reading from files, and writing to files.

file pointer

A pointer that specifies the next byte to be read or the location to receive the next byte written in a file.

file scope

Or global scope. The degree of visibility of an identifier (C/C++ name) when it is declared outside all blocks or classes. The identifier is accessible anywhere in the translation unit after its declaration. See also class scope, function scope, function-prototype scope, local scope, external linkage.

file status

Information on whether the file exists, its creation and modification dates and times, its logical size in bytes, its attributes, and its path. See also file time.

file time

A 64-bit value that gives the file creation time, last access time, or last write time, as represented by the number of 100-nanosecond intervals that have elapsed since January 1, 1601. Windows records each file time in coordinated universal time (UTC) format. See also file status, system time.

File Transfer Protocol

A method of retrieving files to your home directory or directly to your computer using TCP/IP. Many Internet sites have established publicly accessible repositories of materials that can be obtained using FTP with the account name anonymous. Thus, these sites are called anonymous ftp servers.

filename extension

In the MS-DOS 8.3 filename convention, an optional period (.) followed by up to three characters that can be appended to the eight-character base filename. See also base name.

final class

In Java, a class that can have no subclasses.

final method

In Java, a method that cannot be overridden.

final variable

In Java, a variable whose value cannot be changed. Corresponds to the C++ constant.

firewall

Or proxy server. A system or combination of systems that enforces a one-way barrier between two or more networks, usually used for security purposes. Firewalls accomplish all communication between the network and outside.

fixup

1 Or relocation information. Information generated by the linker for addresses that cannot be determined at link time. For example, the linker creates fixups for addresses within the executable that are relative to the executable's base address and for addresses of dynamically loaded DLLs. The Windows loader uses this information to resolve these addresses when the program or DLL is loaded.

2 A record (FIXUPP) generated by the assembler for each address it cannot determine (for example, addresses of external symbols). The fixup contains the information the linker will need to determine the address.

flag

Broadly, a marker of some type used by a computer in processing or interpreting information. Such a signal indicates the existence or status of a particular condition. Depending on its use, a flag can be code, embedded in data, that identifies some condition, such as the beginning or end of a word or a message, or it can be one or more bits set internally by hardware or software to indicate an event of some type, such as an error or the result of comparing two values.

floating toolbar

A toolbar that can appear anywhere on the user's display and is always on top of all other windows. Its size or position can be modified when floating. See also dockable toolbar, docked toolbar.

floating type

A general category of arithmetic data types that are capable of storing a floating-point value (a number that may have a fractional part). The data types **float**, **double**, and **long double** are floating types. See also integral type.

floating-point function

A function that returns a value defined as a floating-point type—that is, a **float**, **double**, or **long double**. See also floating-point number.

floating-point number

Or real number. A value that may contain an integer component, a fractional component, or both. In C/C++, the data types **float**, **double**, and **long double** can store floating-point numbers.

flow control

In data communications, a mechanism to prevent the sender from transmitting data faster than the receiver can handle the incoming data. Flow control usually depends on a set of protocols established at the beginning of the transmission session that define how and when a sender may transmit.

focus

A temporary property of a user-interface object, such as a window, view, dialog box, or button, that permits the object to receive keyboard input from the user. The focus is usually conveyed through highlighting. See also top-level window.

font mapper

In Windows, an operating system component used to find the physical font that most closely matches a specified logical font. The font mapper uses an internal algorithm that compares the attributes of the requested logical font against the attributes of available physical fonts. This mapping occurs when the font is actually used for the first time. See also logical font, physical font.

font resource

A group of individual fonts representing characters in a given character set that have various combinations of heights, widths, and pitches. The Windows operating system maintains a font table containing all the fonts that applications can use. You can load font resources and add the fonts in each resource to this table by using the **AddFontResource** Windows function.

foreground color

The color that is currently selected for drawing or displaying text on screen. In monochrome displays, the foreground color is the color of a bitmap or other graphic. See also background color.

foreground window

Or active window. The window with which the user is currently working. The foreground window is identified by color changes to the title bar and border. See also topmost window, top-level window, focus.

form view

A program window whose client area contains dialog-box controls to permit entering, viewing, or altering data, generally in a form-based data-access application.

form-based **application**

An application whose user interface is based on a form containing controls in which a user examines, enters, or edits data.

formal argument

Or formal parameter. An argument that is declared in the function header and used in the body of a function. Calling functions pass values to called functions in actual arguments. The called function accesses the values using its corresponding formal arguments.

formal parameter list

The parameters specified in a particular method or function definition. Contrasts with the actual parameter list, which appears in the declaration.

format

1 As a noun, the structure or appearance of a unit of data, such as a file, fields in a database record, a cell in a spreadsheet, or the text in a word-processing document.

2 As a verb, to change the appearance or organization of the selected material. To format a disk is to prepare a disk for use by organizing its storage space into a collection of data compartments, each capable of being addressed by the operating system.

format string

A string that can contain specifications for various kinds of type formats as well as literal characters. For example, in the C statement

```
printf( Total Expenses:  $%.2f \n,  Sum );
```

the format string (enclosed in double quotation marks) contains the literal string Total Expenses: \$, the formatting characters `%.2f` to print the value of `Sum` as a decimal number with two digits to the right of the decimal point, and the `\n` character to begin a new line.

formatting rectangle

In Windows, a construct for formatting the text displayed in the window rectangle. An application can make the formatting rectangle larger than the window rectangle (limiting the visibility of the edit control's text) or smaller than the window rectangle (thereby creating extra white space around the text).

forward reference

In C/C++, a reference to a class, variable, or function that has been declared but not yet defined.

frame window

In MFC, the window that coordinates the interactions of the application with a document and its view. The frame window provides a visible frame around a view, with an optional status bar and standard window controls such as a control menu, buttons to minimize and maximize the window, and controls for resizing the window. The frame window is responsible for managing the layout of its child windows and other client-area elements such as control bars and views. The frame window also forwards commands to its views and can respond to notification messages from control windows. In OLE, a frame window is the outermost main window where the container application's main menu resides. See also main frame window.

framework

See application framework.

Frequently Asked Questions

Documents that list and answer the most common questions on a particular subject.

friend

A keyword used within a class declaration to specify that a function or another class has access to the private and protected members of the first class. The function or class specified with the **friend** keyword is considered a friend of the first class.

FTP

A method of retrieving files to your home directory or directly to your computer using TCP/IP. Many Internet sites have established publicly accessible repositories of materials that can be obtained using FTP with the account name anonymous. Thus, these sites are called anonymous ftp servers.

full link

A non-incremental build of program files in which an incremental status (.ILK) file is generated. The incremental status file has the same base name as the executable (.EXE) file or dynamic-link library (.DLL) file targeted by the link, and a .ILK filename extension. During subsequent incremental builds, the linker uses and updates the incremental status file. See also incremental link.

full-server application

In OLE, an application that can be run either as a stand-alone application or launched by a container application. A full-server application can store documents as files on disk and supports both embedding and linking. See also mini-server application, server application.

fully qualified path

Or absolute path. The location of a file or directory on a volume, including a drive letter and any intervening directory names. See also relative path.

function

- 1** A block of code, consisting of a return type, function name, optional parameters, and statements, that performs one or more specific tasks within the source program and returns a value to the caller.
- 2** The purpose of or the action carried out by a program, routine, or other object.

function body

The portion of a function definition that contains the declarations of its local variables and executable statements. See also function declaration.

function call

A postfix expression followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the actual arguments to the function.

function counting

A run-time analysis of a program in which the profiler records how many times each function was called, which is its hit count. See also function profiling, function timing, function coverage, line counting.

function coverage

A run-time analysis of a program in which the profiler reports whether a function was called. This analysis shows which sections of code (functions) are not being executed. See also function counting, function profiling, function timing, line coverage.

function declaration

A statement consisting of a return type, followed by the function name, followed by a list of the names and types of formal parameters enclosed in parentheses. In C++, a function must be declared before it can be called.

function definition

Specifies the name of the function, the types and number of parameters it expects to receive, and its return type. A function definition also includes a function body with declarations of the function's local variables and the statements that determine what the function does. Function definitions differ from function declarations in that they supply function bodies—the code that makes up the function.

function overloading

In C++, specifying more than one function of the same name but with different parameters, in the same scope. These functions, called overloaded functions, enable programmers to supply different semantics for a function, depending on the types and number of arguments. For example an overloaded function could be called `print`, regardless of whether it printed a single string, a list of integers, or all the data members of a class. The arguments supplied in the function call determine which `print` function is called. See also function overriding.

function overriding

In C++, redefining a member function of a base class from within a derived class. An overriding function has exactly the same name, parameters, and return type as the overridden function. See also function overloading.

function profiling

A run-time analysis of code execution by function, in which the profiler detects inefficiencies by counting and timing functions. See also function counting, function coverage, function timing, line profiling.

function prototype

Names the function and its parameters (if any), and provides type information for the return value and parameters (if any). See also formal parameter list.

function scope

In C/C++, the degree of visibility of a label. Labels are accessible only within the function where they are declared. See also class scope, file scope, function-prototype scope, local scope.

function set

One or more functions or data objects that can be exported by a shared library so that they are available to other programs.

function template

- 1 A mechanism for specifying a set of functions that are based on the same code but act on different types or classes. When a templated function is first called for each type, the compiler creates an instantiation, a specialized version of the templated function for the type.
- 2 A complete member function definition with an empty function body that ClassWizard writes in the source files that contain the class of which the member is a function.

function timing

A run-time analysis of a program in which the profiler records how many times each function was called (the hit count) as well as how much time was spent in each function and any called functions. See also function counting, function coverage, function profiling.

function-prototype scope

Or prototype scope. In C++, the degree of visibility of an identifier when it is declared within a function prototype. The identifier is accessible only to the function declarator (delimited by parentheses). Such identifiers are effectively comments, to make the parameters of the prototyped function readily apparent to a reader. See also class scope, file scope, function scope, local scope.

functionality

The features, operations, functions, or capability supported by a program or program component. Some examples of common functionality are serialization, in-place activation, OLE, and ODBC.

fundamental type

A data type that is built into the language. In C/C++, fundamental types can be divided into three categories: integral, floating, and void. Integral types are capable of handling whole numbers. Floating types are capable of specifying values that have fractional parts. The **void** type describes an empty set of values.

garbage collector

A process that periodically frees the memory used by objects that are no longer needed.

gateway

A host computer that connects networks that communicate using different protocols. For example, a gateway connects a company's local area network to the Internet.

GDI

An executable program that processes graphical function calls from a Windows-based application and passes those calls to the appropriate device driver, which performs the hardware-specific functions that generate output. By acting as a buffer between applications and output devices, GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device.

general protection fault

An exception that is raised when an application attempts to read from or write to an area of memory that is not owned by the application. Usually, an application must be terminated after a general protection fault.

GIF

A form of graphics compression. See also Joint Photographic Experts Group (JPEG).

global

Universal, in the sense of being related to an entire file, document, program, or other entity. For example, a global variable is one that is accessible from anywhere in the program.

global variable

A variable that is accessible from anywhere in a program. A global variable has storage and maintains a value throughout the program's execution. See also file scope.

globally unique identifier

See universally unique identifier.

glyph

The bitmap, collection of points, or collection of graphic commands that define a single character or symbol in a font. See also [bitmap](#).

gopher

A client/server application that allows the user to browse large amounts of information. It presents the information to the user in a menu format. When capitalized, it refers to the original Gopher server developed at the University of Minnesota.

GPF

An exception that is raised when an application attempts to read from or write to an area of memory that is not owned by the application. Usually, an application must be terminated after a general protection fault.

graphic object

One of the drawing tools, such as a pen, brush, bitmap, palette, and so on, that Windows provides for use in device contexts.

graphics device interface

An executable program that processes graphical function calls from a Windows-based application and passes those calls to the appropriate device driver, which performs the hardware-specific functions that generate output. By acting as a buffer between applications and output devices, GDI presents a device-independent view of the world for the application while interacting in a device-dependent format with the device.

Graphics Interchange Format

A form of graphics compression. See also Joint Photographic Experts Group (JPEG).

group-box control

A labeled rectangle used to define a group of related controls (usually check boxes or radio buttons) in a dialog box. The group box itself does not set any options; it is simply a graphical device used to improve the usability of complex dialog boxes. See also radio group.

guarded body of code

A set of one or more statements for which an exception or termination handler provides protection.

GUID

See UUID.

handler

- 1 In OLE, a DLL that resides in a client process and performs some tasks on behalf of the server, while delegating other tasks back to the server itself.
- 2 In general, a routine that manages a common and relatively simple condition or operation, such as error recovery or data movement. See also message handler.

hash value

The value of a key that has been numerically manipulated to directly calculate either the location of its associated record in a table or the starting point for a search for the associated record. If the key value is a character string, each possible character is assigned a numeric code to permit the numerical manipulation. The manipulation performed on the key value is known as the hashing function.

head

- 1** The beginning of an item, such as a list, nonscrolling region, string, or transmission queue.
- 2** In hardware, the read/write mechanism in a disk drive or magnetic tape drive, or the printing mechanism in a printer.

header control

In MFC, a window usually positioned above columns of text or numbers. The header control contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column.

header file

An external source file, identified at the beginning of a program, that contains commonly used data types and variables used by functions in the program. The **#include** directive is used to tell the compiler to insert the contents of a header file into the program. See also C++ header file.

heap

- 1** Or free store. A portion of memory reserved for a program to use for the temporary storage of data structures whose existence or size cannot be determined until the program is running. The program can request free memory from the heap to hold such elements, use it as necessary, and later free the memory.
- 2** In sorting, a partially ordered complete binary tree.

heap allocation

Reserving memory for the needs of the program. See also heap, stack.

Help context

A string and a number (Help context ID) that an application passes during a call to Windows Help in order to locate and display a Help topic. See also Help map file, Help project file.

Help file

A file that contains text and graphics needed to communicate online information about an application. Each help file contains one or more topics a user can select by clicking hot spots, using the keyword search, or browsing through topics. Help files have a .HLP filename extension. See also Help topic.

Help map file

A file that defines Help context IDs corresponding to the IDs of dialog boxes, menu commands, and other resources in an application. The AppWizard file MAKEHELP.BAT calls the MAKEHM tool to generate this file from the contents of a RESOURCE.H file. The Help map file has a .HM filename extension. See also Help project file.

Help project file

A project file that controls how the Windows Help Compiler creates a Help (.HLP) file from topic files. The Microsoft Help Workshop is used to create a Help project file. The filename extension of a Help project file is .HPJ.

Help topic

The primary unit of information in a Help (.HLP) file. A topic is a self-contained body of text and graphics, similar to a page in a book. Unlike a page, however, a topic can hold as much information as you require. If there is more information in a topic than the Help window can display, scroll bars appear to let the user scroll through the information.

hexadecimal

Or hex. The base-16 counting system, whose digits are 0 through F. The letters A through F represent the decimal numbers 10 through 15. Two hex digits are needed to represent 1 byte. See also binary.

hidden text

In a document file, special characters that are not normally printed or displayed on the screen. For example, in a rich-text format (.RTF) file for a Help topic, the following hidden text string generates a jump to another Help topic:

```
Jump textcontextstring@d:\path\file.hlp
```


high order

In a group of bits or bytes, the one that carries the most weight or significance. Within a byte, the leftmost bit is the high-order bit. Within a group of bytes, the byte-ordering convention (big-endian or little-endian) determines the position of the high-order byte.

high-performance file system

A fixed-disk file system that organizes the disk into volumes, rather than partitions and logical drives. HPFS supports long, mixed-case filenames and extended attributes, and implements several levels of caching, for improved operating-system performance. The IBM OS/2 operating system supports HPFS. See also file allocation table file system, New Technology file system (NTFS).

highlighting

Altering the appearance of displayed characters as a means of calling attention to them—for example, by displaying them with higher intensity or by using reverse video (dark on light instead of light on dark, or vice versa). Highlighting is often used to denote characters to be deleted, copied, or otherwise acted upon.

hit count

The number of times a function or line of source code is called (or hit), as reported by the profiler when analyzing a program at run time. See also function profiling, line counting, line profiling.

hit-test code

A code that determines the location of the cursor. For example, HTLEFT means the cursor is in the left border of the window; HTMAXBUTTON means the cursor is in a Maximize button.

home page

The main page of a Web site as generated by the developer of the site. Although the home page is often the first page a visitor to the site sees, it is not the same as start page.

hook

- 1** A point in the Windows message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of messages before they reach the target window procedure.
- 2** More generally, a location in a routine or program at which the programmer can connect or insert other routines for the purpose of debugging or enhancing functionality.

hook procedure

An application-installed procedure that monitors the system for events associated with either a specific thread or all threads in the system. For example, the hook code WH_GETMESSAGE installs a hook procedure that monitors messages posted to a message queue. See also callback function.

hot key

- 1** In Windows, an application-defined key combination used to obtain high-priority keyboard input from the user. For example, an application can have a hot key that allows the user to cancel a lengthy operation.
- 2** A key combination that the user can create to perform an action quickly. See also accelerator key.

hot spot

The pixel in a cursor that marks the exact screen location affected by a mouse action, such as a button click. Mouse messages include the coordinates of the hot spot.

hourglass cursor

A cursor displayed in the form of an hourglass that indicates that the program is performing a lengthy task. The cursor can be displayed as a large hourglass, or as a smaller hourglass with an arrow.

HPFS

A fixed-disk file system that organizes the disk into volumes, rather than partitions and logical drives. HPFS supports long, mixed-case filenames and extended attributes, and implements several levels of caching, for improved operating-system performance. The IBM OS/2 operating system supports HPFS. See also file allocation table file system, New Technology file system (NTFS).

HTML

A markup language derived from SGML. Used to create a text document with formatting specifications that tells a software browser how to display the page or pages included in the document.

HTTP

The Internet protocol used by World Wide Web browsers and servers to exchange information. The protocol makes it possible for a user to use a client program to enter a URL (or click a hyperlink) and retrieve text, graphics, sound, and other digital information from a Web server. HTTP defines a set of commands and uses ASCII text strings for a command language. An HTTP transaction consists of a connection, a request, a response, and a close.

HTTP server

Or Web server. A server that runs a Hypertext Transfer Protocol service or application. This protocol defines the procedures used when connecting a Web browser to a Web server.

hyperlink

The means used to jump to another Web page. It consists of both the display text the user sees and the URL of the reference. See also hypertext link.

hyperlink address

The path to an object, document, or page. A hyperlink address can be a URL or a UNC network path. Can also contain display text or sublocation information (for example, a database object, Word bookmark, or Microsoft Excel cell range to which the address points).

hyperlink base

A partial path based on a user-defined path. Use a hyperlink base if you want to move either the file that contains the hyperlink or the destination file separately and still maintain the hyperlink.

hypertext link

In a Web document, this is a reference to another Web document. The user does not actually see a URL in the document; instead, a highlighted reference contains a pointer to a link. The user clicks on the highlighted reference and the desired Web page is automatically retrieved. See also hyperlink.

Hypertext Markup Language

A markup language derived from SGML. Used to create a text document with formatting specifications that tells a software browser how to display the page or pages included in the document.

hypertext reference

Attribute of the HTML anchor element that identifies the anchor as a hyperlink. Its value determines the destination of the hyperlink.

Hypertext Transfer Protocol

The Internet protocol used by World Wide Web browsers and servers to exchange information. The protocol makes it possible for a user to use a client program to enter a URL (or click a hyperlink) and retrieve text, graphics, sound, and other digital information from a Web server. HTTP defines a set of commands and uses ASCII text strings for a command language. An HTTP transaction consists of a connection, a request, a response, and a close.

ICMP

An extension to the Internet Protocol that allows for the generation of error messages, test packets and informational messages related to IP.

icon file

In Windows, a file that contains a bitmap of an icon. Icon files usually have a .ICO filename extension.

icon resource

An icon that is stored in an application's resource-definition file. At run time, an application can call the **LoadIcon** Windows function to retrieve the handle of the icon. Icon resources can be used to avoid device dependence, simplify localization, and enable applications to share icon shapes.

idle state

The state of a modal dialog box or menu when it has finished processing a message and it has no more messages waiting in its active message queue.

idle time

The period during which the application has an empty message queue. Idle time permits the processing of background tasks.

IEEE

A professional organization of computer hardware and software engineers. The institute has developed standards for many aspects of computer technology, such as network connectivity, and formats for representing floating-point numbers.

IETF

The primary working body developing standards for the Internet.

image list

In MFC, a collection of same-sized images contained in a single, wide bitmap. Image lists are used to efficiently manage large sets of icons or bitmaps.

immediate rendering

In a data-transfer operation, making data immediately available to an application, versus making it available when requested (delayed rendering).

impersonation token

In Windows NT, an access token that has been created to capture the security information of a client process, allowing a server to impersonate the client process in security operations. See also primary token, privilege, security identifier (SID).

implementation

A description of the data structure used to represent an object's core state, definitions of the methods that can access the data structure, and information about the intended type of the object. Can also refer to a single function, where the function definition provides the implementation for that function.

implementation file

In MFC, the source code file (usually a C++ source file with a .CPP or .CXX filename extension) that contains a single class definition along with the code that implements that class's member functions. See also interface file.

implementation-defined

Behavior that depends on the implementation, with the range of possible behaviors often delineated by a standard.

import file

A file that describes the functions and data to be exported when an export (.EXP) file is referenced by a linked program. The Incremental Linker (LINK.EXE) uses the export file to build a program that contains exports (usually a dynamic-link library), and it uses the import library to resolve references to those exports in other programs. Import library files usually have a .LIB filename extension. See also library file.

imported function

One called inside another executing entity, such as a DLL or executable file. Functions can be imported either through use of a .LIB import library, or by using the __declspec (dllimport) storage-class modifier, a Microsoft-specific extension to the C language. This modifier explicitly defines the client's interface to other services. See also exported function.

in-memory file

A file that behaves like a disk file except that its bytes are stored in RAM. An in-memory file is a useful means of transferring raw bytes or serialized objects between independent processes.

in-place activation

Or in-place editing, visual editing. The ability to activate an object within the context of its container document, as opposed to opening it in a separate window.

increment (++) operator

An operator that specifies that its integral or floating-point operand be increased by the integer value 1. An incremented pointer points to the next object. In the prefix form (`++i`), the increment takes place before the value is used in expression evaluation; in the postfix form (`i++`), the increment takes place after the value is used in expression evaluation. See also decrement (`--`) operator.

incremental link

The process of rebuilding only those program files that have a timestamp that is more recent than the last link, or updating only those parts of the program that have changed. An incrementally linked program is functionally equivalent to a program that is linked nonincrementally, but it differs from a nonincremental program in that it is prepared for subsequent incremental links and is larger as a result. See also incremental status file, full link.

incremental status file

A state file generated to hold status information for later incremental links of the program. The file has the same base name as the executable file or dynamic-link library and the filename extension .ILK. The incremental status file is created the first time the Incremental Linker (LINK.EXE) runs in incremental mode. LINK updates the file during subsequent incremental builds. LINK is the only tool that uses the .ILK file. See also incremental link.

Indexed Sequential Access Method

Pronounced EYE-sam. A scheme for decreasing the time necessary to locate a data record within a large database, given a unique key for the record. The key is the field in the record used to reference the record.

indirect memory operand

In an assembly-language instruction, a memory operand whose value is treated as an address that points to the location of the desired data.

indirection (*) operator

Or dereferencing operator. In C/C++, a unary operator (*) that accesses a value indirectly, through a pointer. The operand must be a pointer value. The result of the operation is the value stored at the address to which the operand points. If the operand points to a function, the result is a function designator. If it points to a storage location, the result is an l-value designating the storage location. See also address-of (&) operator.

infix notation

Placing of operators between operands, as in $(x+y)*z$. C++ and Java binary operators use infix notation exclusively. Compare prefix notation, postfix notation.

inheritance

In object-oriented programming, a method for deriving new classes from existing classes. The derived class inherits the description of its base class(es), but can be extended by adding new member variables and functions and by using virtual functions. A class can inherit from a single base class (single inheritance) or from any number of direct base classes (multiple inheritance). A class derived using multiple inheritance has the attributes of all of its base classes.

inheritance hierarchy

A classification of items in which each item except the top one (known as the root) is a specialized form of the item above it. Each item can have one or more items below it in the hierarchy. In the Java class hierarchy, the root is the **java.lang.Object** class. In MFC, the root is the **CObject** class. See also root.

initialization file

In Windows, a file that an application uses to store information that otherwise would be lost when the application closes. Initialization files typically contain information such as user preferences for the configuration of the application. Initialization files usually have a .INI filename extension.

initializer

The portion of a declarator that specifies an initial value for an object or a variable.

inline assembler

1 A feature of the Visual C++ compiler that allows the use of assembly-language instructions in C/C++ source programs without extra assembly and link steps. The inline assembler is built into the compiler; therefore, a separate assembler is not needed. Inline assembly code can use any C/C++ variable or function name that is in scope.

2 Assembly-language code that is inserted into C/C++ source code. The **_asm** keyword preceding an assembly-language statement or block of statements invokes the inline assembler at compile time.

inline file

A file that contains text specified in the makefile. The name of file can be used in commands as input (for example, a LINK command file), or it can pass commands to the operating system. The file is created on disk when a command that creates the file is run.

inline function

- 1** In C++, a function defined in the body of a class declaration. Inline functions are typically one- or two-line functions used to return information about an object's state.
- 2** A function whose declaration is preceded by the keyword **inline**, instructing the compiler to replace calls to that function with the code of the function body. This substitution occurs only at the compiler's discretion. For example, the compiler does not expand a function inline if its address is taken or if it is too large to expand inline.

insertion (<<) operator

Or output operator, put-to operator. In C++, the left-shift operator, overloaded (in the iostream library) to provide formatted output. The left operand must be one of the predefined output streams (**cout** or **cerr**) and the right operand can be any valid C++ expression. See also extraction (>>) operator, overloaded operator.

inside-out

In OLE, a model for in-place objects (for example, windows) that the user can activate with a single click. An inside-out object remains visible to the user when its user interface is deactivated. See also outside-in.

instance

- 1** In general, an object of a particular class.
- 2** In Java, an instance of a class is created using the **new** operator followed by the class name.
- 3** In C++, an instance can be created by defining it on the stack. In this case, the **new** keyword is not used. You can also instantiate an OLE object by calling the OLE API **CoCreateInstance**.

instance handle

A handle that Windows assigns to each copy of a loaded application or dynamic-link library (DLL) in a multitasking system. Every window class requires an instance handle to identify the application or DLL that registered the class. See also task handle.

instance method

Any method that can be invoked using an instance of a class, but not using the class name.

Instance methods are defined in class definitions. Called nonstatic member functions in C++. See also class method.

instance variable

Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Called member variables in C++. See also class variable.

Institute of Electrical and Electronics Engineers

A professional organization of computer hardware and software engineers. The institute has developed standards for many aspects of computer technology, such as network connectivity, and formats for representing floating-point numbers.

integral type

A general category of arithmetic data types that are capable of storing an integer (whole number). The data types **char**, **short**, **int**, **long**, and **enum** are integral types. The optional keywords **signed** and **unsigned** can precede or follow any of the integral types except **enum**, or these keywords can also be used alone as type specifiers, in which case they are understood as **signed int** and **unsigned int**, respectively. See also floating type.

Integrated Services Digital Network

A type of digital communications service offered by telephone companies. It can carry data, voice, and video over specially conditioned high-speed telephone lines delivering two 64,000 bps bearer channels and one 16,000 bps data signaling channel.

interface

- 1** In the Component Object Model, a set of related functions; a description of an abstract type.
- 2** In Java, a group of methods that can be implemented by several classes, regardless of where the classes are in the class hierarchy.
- 3** An IDL keyword used by the MIDL compiler for generating interface declarations.

interface file

In MFC, the source header (.H) file that contains a single class declaration and any other information needed to use the class. See also implementation file.

internal linkage

Specifies that the names of objects and functions should refer only to program elements inside their own translation units. The names are not shared with other translation units. The keyword **static** before a name declaration ensures internal linkage. See also external linkage.

Internet

A collection of computer networks that connects millions of computers around the world.

Internet Control Message Protocol

An extension to the Internet Protocol that allows for the generation of error messages, test packets and informational messages related to IP.

Internet Engineering Task Force

The primary working body developing standards for the Internet.

Internet Protocol

The basic protocol of the Internet. It enables the delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability.

Internet Relay Chat

A multi-user system where people convene on channels (a virtual place, usually with a topic of conversation) to talk in groups or privately on the Internet.

Internet server application

Internet server extension DLL.

Internet Server Application Programming Interface

A set of functions for Internet servers, such as a Windows NT Server running Microsoft Internet Information Server (IIS).

Internet server extension DLL

Or Internet server application (ISA). A DLL that can be loaded and called by some HTTP servers. Used to enhance the capabilities of applications that extend a Web server.

Internet server extensions

An application that processes server requests, including Common Gateway Interface (CGI) and ISAPI applications.

Internet server filter

A routine that receives notification of server events such as URL mapping and logon requests, and filters, examines, or changes data to and from the browser or Web server.

Internet Service Provider

A company that provides access to end users of the Internet, as opposed to Network Service Providers (NSP's).

interpreter

A program that can execute code that is not native to the machine it is being run on. Java programs are often run by an interpreter that decodes the bytecodes that make up the program. See also JIT compiler.

intranet

A network within an organization, usually connected to the Internet via a firewall, that uses protocols such as HTTP or FTP to enhance productivity and share information.

intrinsic function

1 In certain high-level programming languages, such as FORTRAN, a function that is part of the language. The compiler automatically links intrinsic functions to the program without any additional effort on the programmer's part. Programs that use intrinsic functions are faster because they do not have the overhead of function calls, but they may be larger due to the additional code generated.

2 In Microsoft C++, a library function, such as **strcmp** or **strcpy**, that has an intrinsic form. Use of the `#pragma intrinsic` compiler directive generates these functions as inline code rather than as function calls.

IP

The basic protocol of the Internet. It enables the delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability.

IP address

The Internet protocol address which is a 32-bit address assigned to a host. The IP address has a host component and a network component.

IP number

An Internet address that is a unique number consisting of 4 parts separated by dots, sometimes called a dotted quad (for example, 198.204.112.1). Every Internet computer has an IP number and most computers also have one or more domain names that are plain-language substitutes for the dotted quad.

IRC

A multi-user system where people convene on channels (a virtual place, usually with a topic of conversation) to talk in groups or privately on the Internet.

ISA

Internet server extension DLL.

ISAM

Pronounced EYE-sam. A scheme for decreasing the time necessary to locate a data record within a large database, given a unique key for the record. The key is the field in the record used to reference the record.

ISAPI

A set of functions for Internet servers, such as a Windows NT Server running Microsoft Internet Information Server (IIS).

ISAPI filter

An Internet server filter packaged as a dynamic-link library that runs on ISAPI-enabled servers.

ISDN

A type of digital communications service offered by telephone companies. It can carry data, voice, and video over specially conditioned high-speed telephone lines delivering two 64,000 bps bearer channels and one 16,000 bps data signaling channel.

ISO/OSI

International Organization for Standardization/Open Systems Interconnection

ISP

A company that provides access to end users of the Internet, as opposed to Network Service Providers (NSPs).

Java

A programming language similar to C++. Java is currently popular because it is compiled to machine-independent bytecode. This allows programmers to write one kind of program code for all platforms (Macintosh, Windows 3.x, Windows NT, Windows 95, UNIX). Software components written in Java for the Web are called applets. Java applets access code libraries on local clients and can download additional class files from the server.

JIT compiler

Just-In-Time compiler for Java. The JIT compiler takes the Java bytecode (which is machine-independent) and compiles it on demand into native code for the target machine, giving faster execution. Since JIT compilers operate on the client machine, they preserve the platform-independence of the compiled Java program.

join

A database operation that combines records from two or more tables, based on an exact match of key values in these tables. Once a recordset has been created based on a join, it functions as if the records were all in one table.

Joint Photographic Experts Group

An algorithm for graphics file storage, supported by many Web browsers. JPEG was developed for compressing and storing photographic images and is best used for graphics containing many colors, such as scanned photos. Most files that are compressed using JPEG are stored in a file format called JFIF (JPEG File Interchange Format). See also Graphics Interchange Format (.GIF).

JPEG

An algorithm for graphics file storage, supported by many Web browsers. JPEG was developed for compressing and storing photographic images and is best used for graphics containing many colors, such as scanned photos. Most files that are compressed using JPEG are stored in a file format called JFIF (JPEG File Interchange Format). See also Graphics Interchange Format ([.GIF](#)).

jump

A word, phrase, or button in a Help topic that, when clicked, takes the user to another topic screen in the Help system.

jump statement

A statement that either transfers control immediately to another location in the function or returns control from the function. In C++, the jump statements are **break**, **continue**, **return**, and **goto**.

kernel

The core component of the operating system. The kernel schedules activities (threads) for the computer processor to perform, and it handles interrupts and exceptions. If the computer has multiple processors, the kernel synchronizes activity among the processors to optimize performance.

key value

A field or expression used to identify a record; often used as the index field for a database table.

keyboard accelerator

See accelerator key.

keyword

Or reserved word. A word that has special meaning to a program or in a programming language. Reserved words usually include those used for control statements, data declarations, and the like. A reserved word can be used only in certain predefined circumstances; it cannot be used in naming functions, variables, structures, labels, classes, or user-generated tools such as macros.

I-value

The value on the left side of an assignment statement that represents a storage region's memory location. The I-value is an expression that evaluates to a type other than **void** and designates a location in memory (a variable, array, or structure element, for example). See also r-value.

label

- 1** In a C or C++ function body, a unique name followed by a colon (:). Labels can denote statements to which a **goto** statement can branch. In a **switch** statement, the labels preceded by the keyword **case** list values to be compared to the expression at the top of the **switch** statement. See also labeled statement.
- 2** In user-interface design, application-defined text associated with a graphical element such as a button or check box.
- 3** In OLE, text describing an object that is linked or embedded as an icon.

labeled statement

In C/C++, a statement preceded by a label. A labeled statement allows program control to be transferred to it from a **goto** statement or back to the top of a **switch** statement.

language identifier (ID)

A 16-bit value that identifies a language and, where appropriate, the variant of the language being used. A language identifier is a combination of a sublanguage identifier and a primary language identifier. For example, if the primary language identifier specifies English, the sublanguage identifier might specify Australian English. See also locale identifier (LCID).

LCID

A 32-bit value identifying the language and sublanguage for a locale. The locale identifier is used to customize string handling. See also code page, language identifier (ID).

lead byte

In double-byte character sets and multibyte character sets, the first byte of a two-byte character. The lead byte signals that both it and the following byte are to be interpreted as a single character. See also trail byte.

left outer join

A database term that describes the relationship between two tables on which a query is run. If two tables are connected with a left outer join, the resulting recordset contains all records from the table on the left but only those records from the table on the right where the joined fields are equal. See also right outer join.

lexical analyzer

Or lexer. The part of a compiler, interpreter, or other translator that parses character sequences (separated by white space) in the source code into individual tokens. See also parser.

library assert

A macro that identifies program errors during development. The argument given to ASSERT should be chosen so that it holds true only if the program is operating as intended.

library file

A Common Object File Format (COFF) file generated by the Microsoft 32-bit library manager tool, LIB, for standard and import libraries. The default filename extension for these files is .LIB. See also dynamic-link library (.DLL), static-link library.

library version

1 One of the available configurations of a library. For example, a library might have a static version and a dynamic-link version, both of which are available as debug and release versions. The library version used during linking is usually determined by the project type and settings.

2 One of the publicly released versions of a library—for example, Microsoft Foundation Class Library version 4.0.

lifetime

The time during program execution that a variable, function, object, or session exists and is available for use, or the duration of the program itself.

Lightweight Remote Procedure Call

In OLE, a protocol for interprocess communication on a single machine. See also Remote Procedure Call (RPC).

line counting

A run-time analysis of a program in which the profiler reports how many times each line of source code was executed. See also function counting, line coverage, line profiling.

line coverage

A run-time analysis of a program in which the profiler reports which lines of source code were executed at least once. See also function coverage, line counting, line profiling.

line profiling

A run-time analysis of a program in which the profiler reports information about the execution of the lines of source code, including line counting and line coverage. Line profiling is useful for checking the validity of an algorithm. See also function profiling.

line-continuation character

The backslash character (\) when it is placed at the end of a line. The line-continuation character causes the compiler to ignore the following newline character and to treat the next line as a continuation of the current line.

linefeed character

A control character that advances a printer or insertion point to a new line. This character can have a different textual representation on different platforms, but it always has the ASCII value of 10. See also carriage return–linefeed (CR-LF) pair.

link

- 1** The Microsoft 32-Bit Incremental Linker (LINK.EXE). See also linker.
- 2** To combine object files and libraries to form an executable file or dynamic-link library.
- 3** In OLE, a connection between two documents. A link has three properties: the name of its source data, its type (or class, as it is known internally), and its updating basis (either automatic or manual). Also, as a verb, it means to connect two documents with a link.

link time

The period of time required by the linker to combine (link) object files and libraries during compilation of a program, or the point in time when these files are combined. See also linker.

linkage specification

The protocol for linking functions (or procedures) written in different languages. Function calling conventions are affected by the linkage specification selected. An example of a linkage specification is **extern C**. See also external linkage, internal linkage.

linked item

Or linked object. In OLE, an item in a compound document whose data is stored in a separate file rather than in the document's file. A linked item must be edited in a separate window. See also embedded item.

linked list

In programming, a data structure consisting of nodes or elements connected by pointers. A singly linked list has one pointer in each node, pointing to the next node in the list; a doubly linked list has two pointers in each node, pointing to the next and previous nodes. In a circular list, the first and last nodes of the list are linked together.

linker

A utility that combines object files and libraries to create an executable file or dynamic-link library. During the linking process, the linker resolves external references such as a call to a library routine by the program.

linking and embedding

Two methods available in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file. See also Object Linking and Embedding (OLE).

list view control

A Windows Common Control that displays a collection of items each consisting of an icon and a label.

list-box control

In Windows, a child window that contains a list of items that can be selected by the user. List boxes can permit the selection of one item or multiple items. See also combo-box control.

listserv

An Internet application that automatically serves mailing lists by sending electronic newsletters to a stored database of Internet user addresses. Although some listservs are moderated, users can often handle their own subscribe/unsubscribe actions without requiring anyone at the server location to personally handle the transaction.

literal

A value, used in a program statement, that is expressed directly rather than as a named constant or the contents of a variable. For example, in the statements

```
i = 25;  
c = 'a';  
cout << Hello;
```

the values 25, 'a', and Hello are literals. See also manifest constant, string literal.

little-endian

One of two byte-ordering conventions used on different machines. In little-endian addressing, the address points to the least significant byte of the word. Intel 80x86 and DEC RISC computers are little-endian machines. See also big-endian.

load time

The period of time required to place a program's executable files into memory prior to execution, or the point in time when the files are loaded.

loaded state

In OLE, the status of a compound-document object (either a linked or embedded item) whose handler is loaded in memory but whose server is not running. See also running state.

local

- 1** In programming, describes an object, memory location, or variable whose scope or lifetime is limited.
- 2** A Microsoft Windows NT keyword that identifies the local attribute, which can be applied to individual functions or to the interface as a whole.
- 3** In distributed systems, describes an operation that is performed by the current application rather than by another application on the same computer or on a remote computer.
- 4** In networking, describes a device that can be accessed directly rather than by means of a communications line.
- 5** In general, an adjective describing an item or operation that is close at hand or restricted to a particular area.

local class

1 A class declared inside a block.

2 Or application local class. In Windows, any window class that an application registers for its exclusive use. Although an application can register any number of local classes, most applications register only one. This window class supports the window procedure of the application's main window. Windows destroys a local class when the application that registered it closes.

local machine

Or local computer. A computer that is accessed directly by the user rather than through an intermediate computer using a communications device. The opposite of a remote machine.

local object

- 1 In C++, an object created inside a function or smaller enclosing block. A local object has scope only within the block in which it is created.
- 2 In OLE, an object (that is not an embedded object) inside a container. Sometimes referred to as a cross-process object.

local **scope**

In C++, the degree of visibility afforded to a name (a variable, for example) when it is declared within a block of code. The name is accessible only from the point of declaration to the end of the block in which it is declared. See also class scope, file scope, function scope, function-prototype scope.

local variable

Or automatic variable. A variable declared within a function body, or smaller enclosing block, and not declared as static, that exists only within the scope of the enclosing curly braces (`{ }`). If a local variable has an initializer, it is initialized every time it is created and its contents are undefined when the function returns.

locale

The national and cultural environment in which a system or program is running. The locale determines the language used for messages and menus, the sorting order of strings, the keyboard layout, and date and time formatting conventions. See also code page, locale identifier (LCID).

locale identifier

A 32-bit value identifying the language and sublanguage for a locale. The locale identifier is used to customize string handling. See also code page, language identifier (ID).

locale-specific

Behavior that depends on local conventions of nationality, culture, and language, such as language, money format, and date format.

locking mode

A strategy for locking records in a recordset during update. A record is locked when it is read-only to all users but the one currently entering data in it. See also optimistic locking, pessimistic locking.

logical brush

An ideal description of a brush bitmap. A logical brush describes all the attributes (style, color, and so on) specified by the application that created it, although some may not be representable on available output devices. Windows also provides seven predefined logical stock brushes. See also brush, physical brush.

logical color palette

An array of colors created by an application that can be selected into a device context and used for graphics output. A logical color is an individual color in the **PALETTE array**. An application using a logical color palette can pass a **COLORREF** value, instead of an explicit red, green, blue (RGB) value, to GDI functions that expect a color. See also color palette, device context (DC).

logical coordinate system

A conceptual coordinate system that a program uses when giving drawing commands. Windows converts logical coordinates to the output device's physical coordinates when it renders the image.

logical font

An ideal description of a font. The font itself is described by the attributes (height, width, orientation, and so on) defined for it by an application. Windows also provides six predefined logical stock fonts. Before an application can begin drawing text with a logical font, it must find the closest match from among the physical fonts stored on the device or in the operating system. See also font mapper, physical font.

logical operator

An operator that performs Boolean evaluations on its operands. The logical-NOT (!) operator produces the value 0 if its operand is true (nonzero) and the value 1 if its operand is false (0). The logical-AND (&&) operator produces the value 1 if both operands have nonzero values; otherwise, it produces the value 0. The logical-OR (||) operator produces the value 1 if either of its operands has a nonzero value. See also bitwise operator.

logical shift

Or bitwise shift. A bitwise operation wherein the bits in the first operand are shifted by the number of positions specified by the second operand. The operator specifies the direction the bits are shifted. See also arithmetic shift.

logical unit

A conceptual unit of measure for graphics device interface (GDI) functions. A logical unit is converted to a device, or physical, unit (for example, to a number of pixels or to a distance in inches) when an image is rendered on the output device. By default, GDI considers logical units to be equal to device units, meaning that 1 logical unit equals 1 pixel on the screen.

logical view

A view expressed in terms of logical coordinates. A logical view can change its scaling when it is rendered onto an output device such as a display monitor.

lookup table

A reference table that maps an index or key to a value to be looked up and returned. A lookup table is often used as an alternative to lengthy run-time calculations.

loop

- 1** A set of statements in a program executed repeatedly, either a fixed number of times or until a condition is true or false.
- 2** In an event-driven state machine, an action that returns control to the original state, either directly or through a series of state transitions.

loss of significance

In programming, a potential error caused by presenting an argument to a math function that is either so large or so small that the operation returns a value that has lost most or all of the significant digits of the original argument. See also underflow.

low order

A description applied to the rightmost element in a group—the one that carries the least weight or significance. For example, the low-order bit in a byte would be the rightmost one.

LRPC

In OLE, a protocol for interprocess communication on a single machine. See also Remote Procedure Call (RPC).

luminance

In graphics, the perceived brightness of a surface or pixel. Luminance often refers to as a weighted average of red, green, and blue color values that gives the perceived brightness of the combination. See also red, green, blue (RGB); red, green, blue, alpha (RGBA).

machine code

The ultimate result of the compilation of assembly language or any high-level language, such as C++ sequences of bytes that are loaded and executed by a microprocessor. Machine code is the only language that computers understand; all other programming languages represent ways of structuring human language so that humans can get computers to perform specific tasks.

Macintosh binary resource file

A Macintosh resource file that has been created from a Windows resource script (.RC) and compiled using the Windows Portability Library version of the Windows Resource Compiler (RC.EXE). By default, .RSC is the filename extension for Macintosh binary resource files.

Macintosh Resource Compiler

A command-line tool, MRC.EXE, controlled by a .R resource script file, that compiles an application's Macintosh-specific resources.

macro

- 1** In C and C++, a name defined by the **#define** directive that specifies replacement text for subsequent invocations of the macro in the translation unit.
- 2** In applications, a set of keystrokes and instructions recorded and saved under a short key code or macro name. When the key code is typed or the macro name is used, the program carries out the instructions of the macro.

macro definition

- 1** In a C/C++ source file, the **#define preprocessor directive**, followed by an identifier, followed by an expression or statement. During compilation, every occurrence of the identifier as a token (that is, not in a comment or a literal string) is replaced with the expression.
- 2** In general, a name and a set of instructions that are recorded in a program so that the instructions can be substituted for the name.

macro expansion

In C/C++, the process of replacing a macro name (defined with a **#define** directive) in the source code with the body of the macro during compilation.

main application window

Or main window. In an SDI application, the window that serves as the primary interface between the user and the application.

main frame window

The primary window responsible for coordinating the frame with its view. In an SDI application, the document frame window is also the main frame window. In an MDI application, document windows are child windows displayed in the main frame window. The styles and other characteristics of frame windows are inherited from the main frame-window class.

main message

The main loop in an application that retrieves messages from a message queue and dispatches them to the appropriate window procedures. The main message loop takes over when there is no secondary loop to handle a particular message. Every application for Windows has a main message. See also message queue.

major version number

In a software version number, the number on the left of the decimal point. For example, the major version number of Windows NT version 3.51 is 3. See also minor version number.

makefile

A file that contains all commands, macro definitions, options, and so on to specify how to build the projects in a project workspace. A makefile has the filename extension .MAK and usually has the same base name as the workspace configuration (.MDP) file.

manifest constant

Or symbolic constant. In C/C++, an identifier defined in a **#define** preprocessor directive to represent a constant value. For example, `#define PI 3.14` declares a manifest constant named `PI`. In C++, the keyword **const** can be used in place of the **#define** directive for declaring a constant.

mapfile

A text file that contains information about the program being linked, including the groups in the program and a list of public symbols. The linker names the mapfile with the base name of the program and the filename extension .MAP.

MAPI

A set of functions that is part of the Win32 API that applications use to create, manipulate, transfer, and store various kinds of communications, such as e-mail messages. It provides application development tools for defining message purpose and content, and allows flexible management of stored messages. MAPI also provides a common development interface for creating communication-enabled and communication-aware applications independent of the underlying messaging system.

marshaling

In OLE, the process of packaging and sending interface parameters across process boundaries. See also Remote Procedure Call (RPC).

mask

- 1** A binary value used to selectively screen out or let through certain bits in a data value.
- 2** (verb) An operation used to select certain bits from a data value, by using a logical operator to combine the mask and the data value. For example, the mask 0x3F, when used with the logical-AND **&&** operator, removes (masks off) the two uppermost bits in an 8-bit data value but does not affect the rest of the value.

mask bitmap

A bitmap that selectively lets through or screens out values in another bitmap. This is accomplished by combining the two bitmaps in a bitwise logical operation.

maximized window

An application window enlarged to fill the entire desktop, a document window enlarged to fill the entire application workspace, or a client window enlarged to fill the client area of the client window. See also minimized window.

MBCS

A character set in which each character is represented as either a 1-byte or 2-byte value. The standard ASCII character set is a subset of the MBCS, which is used for languages that require a set of more than 256 characters. See also double-byte character set (DBCS), lead byte, trail byte, Unicode.

MDI

The standard user-interface architecture for Windows-based applications. A multiple document interface application enables the user to work with more than one document at the same time. Each document is displayed within the client area of the application's main window. See also child window, client area, single document interface (SDI).

member

In C/ C++, one of the elements of a structure, union, or (in C++) a class. In a class, both variables and functions are considered members of the class.

member function

In C++, a function declared inside a class definition. A class's member functions are used to get and set data members, display information to the user, and manipulate data according to the needs of the program. See also data member.

member-access control

Part of the functionality of C++ that prevents data from being used in ways that the programmer did not intend. Member-access control is implemented by specifying class members as private, protected, or public.

memory allocation

A response by the operating system to a request from a program for memory. The two basic types of memory allocation are static allocation, in which memory is set aside when the program starts and remains allocated while the program is running, and dynamic allocation, in which memory is allocated and deallocated while the program is running.

memory block

A section of random-access memory temporarily assigned to a program by the operating system.

memory device context

A block of memory that represents a display surface. A memory device context can be used to prepare images in memory before copying them to the actual display surface of the compatible device.

memory leak

A loss of memory resources that occurs when memory is allocated on the heap and never deallocated to make it available for reuse.

memory model

The approach used to address the code and data used in a computer program. The memory model dictates how much memory can be used in a program for code and how much for data. Most computers with a flat address space support only a single memory model. Computers with a segmented address space usually support multiple memory models. For example, in the 16-bit Microsoft C small memory model, code, data, and data arrays can each be a maximum of 64K in total memory.

memory space

See address space.

menu resource

A collection of information that defines the appearance and function of an application menu. This information includes the text strings that appear in an application's menu bar, menu-item identifiers, the arrangement of the menus and status of menu items.

menu-item identifier

Or menu-item ID. The command identifier chosen for a menu command during the creation of a menu item. Menu-item identifiers can be preassigned values (for example, assigned by the Microsoft Foundation Class Library) or values specified by the programmer. **ID_EDIT_CLEAR_ALL** is the preassigned menu-item ID for a Clear All command on the Edit menu.

message

A structure or set of parameters used for communicating information or a request. Messages can be passed between the operating system and an application, different applications, threads within an application, and windows within an application.

message box

A window that displays information to the user. For example, a message box can inform the user of a problem that the application has encountered while carrying out a task.

message filter

A filter that an application uses to retrieve specific messages from the message queue (while ignoring other messages). The filter is a range of message identifiers (specified by a first and last identifier), a window handle, or both.

message handler

In Windows, an object, such as a view or frame window, that provides handler functions to process messages.

message handling

The act of responding to messages received from the operating system. Applications use message-handling functions to process messages.

message loop

A program loop that retrieves messages from a thread's message queue and dispatches them to the appropriate window procedures. See also message queue.

message map

A mechanism to route Windows messages and commands to the windows, documents, views, and other objects in an MFC application. Message maps map Windows messages, commands from menus, toolbar buttons, accelerators and control-notification messages. A collection of macros in a class's source (.CPP) file specify which functions will handle various messages for the class.

message pump

A program loop that retrieves messages from a thread's message queue, translates them, offers them to the dialog manager, informs the MDI manager about them, and dispatches them to the application. See also message queue.

message queue

A repository for window messages awaiting processing by a thread. The system message queue holds mouse and keyboard input waiting to be passed to a thread's message queue. A thread's message queue holds messages waiting to be retrieved by a thread's message loop.

message-driven

Or event-driven. A programming model in which the state of the running program changes in response to user actions and other events. These events cause the operating system to send messages to the part of the application that can handle the event.

message-handler function

In Windows, a function that responds to a message using parameters that have been translated from *wParam* and *lParam* and passed to the function. A message-handler function might track mouse activity or call member functions of the document to update its data.

message-map entry

An individual item in a message-map table that specifies the handler for a particular message. The framework searches the table for each incoming message and calls the handler for it automatically. See also message map.

message-map macro

One of the macros supplied by the Microsoft Foundation Class Library and used in a message map to specify which messages will be handled by which functions. Message-map macros are available to map Windows messages, command messages, and ranges of messages.

Messaging Application Programming Interface

A set of functions that is part of the Win32 API that applications use to create, manipulate, transfer, and store various kinds of communications, such as e-mail messages. It provides application development tools for defining message purpose and content, and allows flexible management of stored messages. MAPI also provides a common development interface for creating communication-enabled and communication-aware applications independent of the underlying messaging system.

metafile

A collection of structures that stores a picture in a device-independent format. A metafile defines an image as coded lines and shapes. Win32 uses enhanced-format metafiles, which have a .EMF filename extension. Windows metafiles, which have a .WMF filename extension, are used in Windows 3.x.

method

In object-oriented programming, a procedure that provides access to an object's data. In C++, public member functions are the equivalent of methods.

metric

- 1** A dimension (width or height) of a display element, such as a cursor.
- 2** One of the attributes of a font as described in a font specification. For example, the character placement and line-spacing metrics enable an application to compute device-independent line breaks that are portable across screens, printers, typesetters, and even platforms.

MFC

A set of C++ classes that encapsulate much of the functionality of applications written for the Microsoft Windows operating systems.

Microsoft Foundation Class Library

A set of C++ classes that encapsulate much of the functionality of applications written for the Microsoft Windows operating systems.

Microsoft Internet Information Server

A Web server integrated into Windows NT Server which provides FTP, HTTP, and gopher services.

MIIS

A Web server integrated into Windows NT Server which provides FTP, HTTP, and gopher services.

MIME

A standard that allows binary data to be published and read on the Internet. The header of a file with binary data contains the MIME type of the data; this informs client programs (Web browsers and mail packages, for instance) that they will need to handle the data some way other than they handle straight text. For example, the header of a Web document containing a JPEG graphic contains the MIME type specific to the JPEG file format. This allows a browser to display the file with its JPEG viewer, if one is present.

mini-server application

A type of server that can only be launched from a container and that only supports embedding, not linking. Objects created by mini-servers are always stored within compound documents, never as individual files. Microsoft Draw is an example of a mini-server. See also full-server application.

minimal rebuild

A feature that permits the build engine to analyze dependencies within your project and to skip compilation of files that it detects are unaffected by a change, even if those files include changed headers.

minimized window

A window that has been shrunk to an icon. See also maximized window.

minor version number

In a software version number, the number on the right of the decimal point. For example, the minor version number of Microsoft Windows NT version 3.51 is 51. See also major version number.

mnemonic **key**

See access key.

mnemonics

In a graphical user interface, underlined letters in the text of menu and dialog-box items. When the menu and dialog-box items are active, the user can select the item by pressing the key that corresponds to the underlined letter. See also access key.

modal

A restrictive or limiting interaction created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to its primary window or to the entire system. A modal dialog box must be closed by the user before the application continues. See also modeless, system modal dialog box.

modal loop

A message-processing loop during which the system retrieves and dispatches messages without allowing an application the opportunity to filter the messages in its main message

modeless

Not restrictive or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with other windows. A modeless dialog box stays on the screen and is available for use at any time but also permits other user activities. See also modal.

modifier

1 In C and C++, a reserved word in a declaration that qualifies the identifier. The storage-class specifier **extern** and the calling-convention keyword __declspec(thread) are examples of modifiers.

2 Generally, anything that limits or restricts meaning, as in a modifier flag.

modifier **key**

A keyboard key that changes the action of normal input when it is pressed. For example, CTRL, ALT, and SHIFT are modifier keys.

module-definition file

A text file that contains one or more statements describing various attributes of an executable module. Module-definition files usually have a .DEF filename extension. See also [dynamic-link library](#) file.

moniker

An object that supports the **IMoniker** interface, which includes the **IPersistStream** interface; thus, monikers can be saved to and loaded from streams. Monikers can be saved to persistent storage, and they support binding.

Mosaic

A World Wide Web browser written by NCSA.

most recently used

A file list, control of which is supported by the **CRecentFileList** class in MFC. Files can be added to or deleted from the MRU file list. The file list can be read from or written to the registry or an .INI file, and the menu displaying the MRU file list is updatable.

mount

To make a volume (a physical disk) accessible to a computer's file system. The volume could be a local drive or one accessed through a network connection. The term usually describes accessing disks in Apple Macintosh and UNIX-based computers.

mouse capture

The act of channeling mouse input to a specific window without regard to the position of the mouse-cursor hot spot.

mouse event

An input event that occurs whenever the user moves the mouse, or presses or releases a mouse button. Windows converts mouse input events into messages and posts them to the appropriate thread's message queue. See also mouse message.

mouse message

A message sent to a window when a mouse event occurs while the cursor is within the borders of the window, or when the window has captured the mouse. Mouse messages are divided into two groups: client-area messages and nonclient-area messages. See also mouse capture.

Moving Picture Experts Group

Compression algorithms for both video and sound. There are multiple standard levels for MPEG (MPEG1, MPEG2, and so on). Movies saved in the MPEG format are highly compressed and consume much less disk space than would normally be required.

MP

The Internet Engineering Task Force (IETF) standard for aggregating multiple ISDN B channels using synchronous PPP framing.

MPEG

Compression algorithms for both video and sound. There are multiple standard levels for MPEG (MPEG1, MPEG2, and so on). Movies saved in the MPEG format are highly compressed and consume much less disk space than would normally be required.

MRC

A command-line tool, MRC.EXE, controlled by a .R resource script file, that compiles an application's Macintosh-specific resources.

MRU

A file list, control of which is supported by the **CRecentFileList** class in MFC. Files can be added to or deleted from the MRU file list. The file list can be read from or written to the registry or an .INI file, and the menu displaying the MRU file list is updatable.

multibyte character set

A character set in which each character is represented as either a 1-byte or 2-byte value. The standard ASCII character set is a subset of the MBCS, which is used for languages that require a set of more than 256 characters. See also double-byte character set (DBCS), lead byte, trail byte, Unicode.

multiline edit control

A text box that displays more than one line of text. In a multiline edit control, data that is too long to fit on one line can either wrap to the next line or extend beyond the right boundary of the box.ox. See also single-line edit control.

Multilink PPP

The Internet Engineering Task Force (IETF) standard for aggregating multiple ISDN B channels using synchronous PPP framing.

multiple document interface

The standard user-interface architecture for Windows-based applications. A multiple document interface application enables the user to work with more than one document at the same time. Each document is displayed within the client area of the application's main window. See also child window, client area, single document interface (SDI).

multiple document types

More than one kind of document—for example, a text document and a graphics document. In an application (such as Microsoft Excel) that supports multiple document types, each document type puts its own menus into the menu bar and changes the main frame window's caption.

multiple-selection list box

A list box in which more than one item can be selected at the same time.

Multipurpose Internet Mail Extensions

A standard that allows binary data to be published and read on the Internet. The header of a file with binary data contains the MIME type of the data; this informs client programs (Web browsers and mail packages, for instance) that they will need to handle the data some way other than they handle straight text. For example, the header of a Web document containing a JPEG graphic contains the MIME type specific to the JPEG file format. This allows a browser to display the file with its JPEG viewer, if one is present.

multithreaded application

An application capable of carrying out multiple, independent paths of execution (or threads) at the same time. For example, an application may have additional threads to handle background or maintenance tasks so that the user doesn't have to wait for a task to complete before continuing to work with the application. See also user-interface thread, worker thread.

mutex object

In interprocess communication, a synchronization object whose state is signaled when it is not owned by a thread and nonsignaled when it is owned. Only one thread at a time can own a mutex.

name decoration

The process during compilation of a C++ function definition or prototype in which the compiler creates a string that uniquely identifies a function. Name decoration is necessary for the compiler and linker to distinguish between several functions that may have the same name. See also decorated name.

National Center for Supercomputer Applications

A research center at the University of Illinois in Urbana-Champaign for developing and implementing a strategy to create, use, and transfer advanced computing and communication tools and information technologies. NCSA developed Mosaic.

native

- 1** An adjective describing a programming element that is specific to and supported by the particular platform on which it is running.
- 2** An adjective describing a programming element that is specific to a particular country.
- 3** In OLE, an adjective describing data that originated in the container document.

NCSA

A research center at the University of Illinois in Urbana-Champaign for developing and implementing a strategy to create, use, and transfer advanced computing and communication tools and information technologies. NCSA developed Mosaic.

nest

To embed one construct (for example, a function) completely within another.

nested class

A class that is declared within the scope of another class. A nested class is available for use within the scope of the enclosing class. To refer to a nested class from a scope other than its immediate enclosing scope, a fully qualified name must be used.

Network Service Provider

A company that provides connectivity to the Internet for ISPs and others requiring high speed connections between their LANs and the Internet.

New Technology file system

A fixed-disk file system that offers security features, including execute-only files and permissions for individual files, support for Unicode and long filenames, and the ability to handle large storage media. See also high-performance file system (HPFS).

newline character (\n)

The character used to mark the end of a line in a text file, or the escape sequence (\n) used to represent this character.

NMAKE

The Microsoft Program Maintenance Utility. NMAKE reads a description file that specifies project-file dependencies and automatically executes the commands needed to update the project when any project file has changed.

nonclient area

The parts of a window that an application does not use when displaying output such as text or graphics. A window's nonclient area consists of the border, menu bar, title bar, scroll bar, Control menu, Minimize button, and Maximize button. See also client area.

nonintegral expression

An expression that evaluates to a data type other than a **char**, **short**, **int**, **long**, or enumerated type.

nonscalar type

See aggregate type.

nonscalar value

A value whose type is a complex data structure such as a structure, class, or array. See also aggregate type.

nonsystem key

A keyboard key that is pressed when the ALT key is not pressed or a keyboard key that is pressed when a window has the keyboard focus.

notification message

A message that a control sends to its parent window when events, such as input from the user, occur.

NSP

A company that provides connectivity to the Internet for ISPs and others requiring high speed connections between their LANs and the Internet.

NT-1

In ISDN, a network terminator composed of a connector that attaches a two-wire ISDN line to a four-wire line so that it can be connected to PCs and terminals.

NTFS

A fixed-disk file system that offers security features, including execute-only files and permissions for individual files, support for Unicode and long filenames, and the ability to handle large storage media. See also high-performance file system (HPFS).

null brush

Or hollow brush. A logical brush created from a bitmap whose color matches the current window background color. See also bitmap.

null pointer

A pointer to nothing, expressed in C++ as the value 0.

null statement

A statement that performs no action, declares nothing, and can be used wherever a statement is expected. In C/C++, a statement that contains only a semicolon (;).

null string

Or empty string, zero-length string. In C/C++, a string that contains no characters. A null string is explicitly initialized with a pair of double quotation marks ().

null terminator

Or string terminator. In C/C++, the null character `\0` at the end of a string.

null-terminated character string

Or ASCIIZ string. A string of characters terminated by a single character, '\0', whose integer value is 0.

object

Generally, an instance of an entity that embodies both specific data and the functions that manipulate it. Specifically in object-oriented programming, an object is an entity that has state, behavior and identity. An object's state consists of its attributes and the attributes' current values. An object's behavior consists of the operations that can be performed on it and the accompanying state changes. An object's identity is what you use to distinguish it from other objects. In contrast, COM objects' behavior is defined by the interfaces it supports. A COM object's state is not explicitly specified, but is implied by its interfaces. A COM object's identity is defined by the ability to use **IUnknown::QueryInterface** to move between interfaces.

object code

Executable code generated by a compiler or an assembler from the source code of a program.
See also object file.

object description language file

In OLE Automation, text files containing a description of an application's interface. Object description language scripts are compiled into type libraries using the MkTypLib tool included with the OLE Software Development Kit.

object file

A file containing object code and/or data generated by a compiler or an assembler from the source code of a program. Object files generated by the Visual C++ compiler have a .OBJ filename extension. See also Common Object File Format (COFF).

object library file

A dynamic-link library with a type library resource. An object library file typically has a .OLB filename extension.

Object Linking and Embedding

Pronounced o-LAY. A way to transfer and share information between applications. Linking and embedding are two methods in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file.

Object Module Format

A specification for the structure of object (.OBJ) files. The Microsoft Visual C++ version 4.0 linker accepts object files that are either COFF or 32-bit OMF. See also Common Object File Format (COFF), EDITBIN.

object-oriented design

Or object-oriented programming. In traditional procedural languages (such as C, Fortran, and Cobol) code and data are separate. In the object-oriented approach, code and data that belong together can be combined into objects. Object-oriented design is further characterized by the use of inheritance (derived classes), polymorphism, encapsulation, and virtual functions (C++) in programming.

ODBC

An open, vendor-neutral interface for database connectivity that provides access to a variety of personal computer, minicomputer, and mainframe systems, including Windows-based systems and the Apple Macintosh. The ODBC interface permits an application developer to develop, compile, and ship an application without targeting a specific database management system (DBMS). Users can add modules called database drivers that link the application to their choice of database management systems.

ODBC cursor library

A dynamic-link library that resides between the ODBC Driver Manager and the drivers and handles scrolling through data.

ODBC driver

A dynamic-link library file that implements ODBC function calls and interacts with a data source. A driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated database management system.

OEM character set

Printable characters used in full-screen MS-DOS sessions for screen display. Characters 32 through 127 are usually the same in the OEM, U.S. ASCII, and Windows character sets. The other characters in the OEM character set (0 through 31 and 128 through 255) correspond to the characters that can be displayed in a full-screen MS-DOS session. These characters are generally different from the Windows characters and may vary from one original equipment manufacturer (OEM) to another.

offset

In relative addressing methods, a number that tells how far from a starting point a particular item is located. For example, in the search for a specific data item stored within a known area (segment) of memory, an offset is used to tell the microprocessor how many bytes past the beginning of the segment the item is located.

OLE

Pronounced o-LAY. A way to transfer and share information between applications. Linking and embedding are two methods in OLE for storing items inside a compound document when those items were created in another application. An embedded item is stored as part of the compound document that contains it. A linked item stores its data in a separate file.

OLE Automation

A way to manipulate an application's objects from outside the application. OLE Automation is typically used to create applications that expose objects to programming tools and macro languages, to create and manipulate one application's objects from another application, or to create tools for accessing and manipulating objects.

OLE Automation object

Or exposed object. An instance of a class that is defined within an application, exposed for access by other applications or programming tools by means of OLE Automation interfaces. See also OLE Automation client, OLE Automation server.

OLE Automation server

An application that exposes programmable objects to other applications, which are called automation clients. Exposing programmable objects enables clients to automate certain functions by directly accessing those objects and using the services they make available. For example, a word processor might expose its spell-checking functionality so that other programs can use it. See also automation client.

OLE item

Or OLE (pronounced o-LAY) object. An object that represents data, created and maintained by a server application, that can be seamlessly incorporated into a document so that the object appears to be a part of the larger document. The result is a compound document made up of the OLE item and a containing document.

OLE system DLLs

Pronounced o-LAY. The means by which OLE containers and OLE servers communicate. The OLE system dynamic-link libraries (DLLs) provide functions that containers and servers call, and the containers and servers provide callback functions that the DLLs call. Using this means of communication, a container doesn't need to know the implementation details of the server application. A container can accept items created by any server without having to define the types of servers with which it can work

OMF

A specification for the structure of object (.OBJ) files. The Microsoft Visual C++ version 4.0 linker accepts object files that are either COFF or 32-bit OMF. See also Common Object File Format (COFF), EDITBIN.

Open Database Connectivity

An open, vendor-neutral interface for database connectivity that provides access to a variety of personal computer, minicomputer, and mainframe systems, including Windows-based systems and the Apple Macintosh. The ODBC interface permits an application developer to develop, compile, and ship an application without targeting a specific database management system (DBMS). Users can add modules called database drivers that link the application to their choice of database management systems.

Open System Interconnection

A model that breaks network management into five functional areas: fault management, configuration management, security management, performance management, and accounting management.

operand

An entity on which an operator acts.

operator

Symbols that specify an evaluation to be performed on one operand (unary operator), two operands (binary operator), or three operands (ternary operator). See also operator precedence.

operator precedence

The relative position of an operator in the hierarchy that determines the order in which expressions are evaluated.

optimistic locking

A recordset locking strategy in which records are left unlocked until explicitly updated. The page containing a record is locked only while the program updates the record, not while a user is editing a record. See also pessimistic locking.

optimization

Compiler fine tuning to increase program performance or reduce program size. Visual C++ provides five optimization options: Default, Disable (Debug), Maximize Speed, Minimize Size, and Customize.

OSI

A model that breaks network management into five functional areas: fault management, configuration management, security management, performance management, and accounting management.

out-of-band data

A logically independent transmission channel associated with each pair of connected stream sockets. Out-of-band data is delivered to the user independently of normal data. The abstraction defines that the out-of-band data facilities must support the reliable delivery of at least one out-of-band message at a time. This message may contain at least 1 byte of data, and at least one message may be pending delivery to the user at any one time.

out-of-memory exception

An error that occurs during the execution of a program when an out-of-memory situation is encountered.

output precision

In specifying a font, defines how closely the output must match the requested font's height, width, character orientation, escapement, and pitch.

outside-in

A model for in-place OLE objects that requires the user to double-click or select a verb from the Edit menu to activate the object. Outside-in objects are hidden from the user when inactive. See also inside-out.

overhang

The amount by which the black part of a glyph extends beyond its character cell on either side. See also ABC width, underhang.

overlapped I/O

See asynchronous operation.

overlapped window

A style of window meant to serve as an application's main window. Other windows can overlap the window's space on the screen.

overloaded operator

In C++, an operator that has been given functionality beyond that which is built into the language. For example, the programmer can overload the arithmetic operators to allow them to use objects of a given class as their operands. Overloading is usually done to make the source code more readable.

overloading

Supplying more than one definition for a given function name or operator name in the same scope. For example, the addition (+) operator can be overloaded to add the values contained in two objects of the same class. See also function overloading.

owned window

A window that has an owner. An owned window always appears in front of its owner window, is hidden when its owner window is minimized, and is destroyed when its owner window is destroyed. See also owner window.

owner window

A window that owns another window, thus affecting aspects of the owned window's appearance and behavior. See also owned window.

owner-draw control

In Windows, a control that is customized for a specific application. Owner-draw controls are similar to predefined controls in that Windows will handle the control's functionality and process input from the mouse and keyboard. However, the programmer is responsible for the appearance of the owner-draw control in its various states.

package

- 1** In VC++ Developer Studio, a dynamic-link library that extends the development environment.
- 2** In Java, a group of classes declared with the **package** keyword. Classes are generally packaged together when they are related in a significant way. In Java, the default level of data access is at the package level. In C++, the default level of data access is at the class level.

packaged function

A function created by the compiler when function-level linking is enabled. A packaged function is visible to the linker in the form of a COMDAT record placed in the object (.OBJ) file by the compiler. Functions that are not packaged can be linked only at the object level.

Packet Internet Groper

A program useful in testing and debugging LAN/WAN troubles. It sends out a packet and expects a specified host to respond back in a specified time frame.

palette

- 1 An array of colors currently available on a device.
- 2 A modeless secondary window that displays a toolbar or another set of controls.

pane

One of the separate areas in a split window, or a rectangular area of the status bar that can be used to display information.

PAP

In ISDN, a type of authentication accomplished by sending user ID/password pairs between two devices.

parameter

Or argument. A value or an expression used with an operator or passed to a subprogram (subroutine, procedure, or function). The program then carries out operations using the arguments. Parameters are also used with templates; such parameters can include classes as well as values and expressions.

parameter data member

In recordsets, a data member that accepts parameter information at run time. Parameter data members can be used in a query to select records from a database based on the value of the parameter obtained or calculated at run time. See also field data member.

parameter passing

The sending of variables as parameters (arguments) to a subroutine, procedure, or function. You can pass parameters by value or by reference. Passing by value copies the variable's value into the subroutine's formal argument. Passing by reference copies the variable's memory address into the subroutine's formal argument. In C++, parameters may be passed by value or by reference. In C, all parameters are passed by value, except for arrays, which are translated into the address of the first member and reference of type, as in `int&ri`. However, you can force a pass by reference by using pointers or references. In Java, parameter passing is by value only for simple types, by reference only for instances.

parent process

In a multitasking environment, a process that initiates one or more processes, called child processes. The parent process defines the environment for child processes. The parent process can suspend or complete its own processing while the child processes continue operating.

parent window

A window that generates one or more child windows.

parse map

Code that allows a **CHttpServer**-derived object in MFC to map client requests to its member functions. MFC provides parse map macros to help map form variables on a Web page to parameters in your function.

parser

The part of a compiler, interpreter, or other translator that accepts tokens from the lexical analyzer and attaches semantics to those tokens.

Pascal calling convention

A calling convention that pushes arguments onto the stack in the order in which they appear in the function call (from left to right). The Pascal calling convention requires that the called routine remove arguments from the stack.

Password Authentication Protocol

In ISDN, a type of authentication accomplished by sending user ID/password pairs between two devices.

path bracket

One or more graphics device interface (GDI) functions embedded between a **BeginPath** and an **EndPath** function. (The **BeginPath** function defines the start of the bracket; the **EndPath** function defines the end.)

pen

In graphics applications, a drawing object used to draw lines and borders.

PERL

A general-purpose scripting language used mainly for server-side HTML programming. It is freely available on a wide variety of platforms.

permanent storage

- 1 A disk medium, such as a hard disk or floppy disk.
- 2 A place to store data, such as a file or database.
- 3 In the Visual C++ documentation, a variable declared with the **static** storage-class specifier.

persistent

Lasting between program sessions, or renewed when a new program session is begun.

pessimistic locking

A recordset locking strategy in which a page is locked once a user begins editing a record on that page. While the page is locked, no other user can change a record on that page. The page remains locked until records are updated or the editing is canceled. See also optimistic locking.

physical brush

A bitmap that Windows uses to paint the interior of filled shapes. A physical brush is a device driver's approximation of a logical brush.

physical device

- 1** The fourth and final coordinate space for most graphics device interface (GDI) drawing operations. (The Win32 API uses four coordinate spaces: world, page, device, and physical device.) The points in this physical device coordinate space can correspond to the desktop, a client area, or a page of printer paper.
- 2** Hardware such as a modem or printer to which data can be written.

physical font

A font stored internally on a device or whose resources have been loaded into the operating system.
See also logical font.

PING

A program useful in testing and debugging LAN/WAN troubles. It sends out a packet and expects a specified host to respond back in a specified time frame.

pipe

A communication conduit with two ends. A process with a handle to one end can communicate through a pipe with a process having a handle to the other end. Pipes can be one way (where one end is read-only and the other end is write-only) or two way (where both ends of the pipe can be used for reading or writing). The Win32 application programming interface (API) provides both anonymous (unnamed) pipes and named pipes.

pixel

The smallest addressable picture element (that is, a single dot) on a display screen or printed page.
See also raster.

placeholder

A symbol in an expression indicating information to be supplied. For example, in the following SQL statement

```
SELECT CourseID, CourseTitle FROM Course WHERE CourseID = ?
```

the question mark (?) is a placeholder for a parameter value supplied at run time.

platform

The hardware and operating system that support an application. Platform sometimes refers to the hardware alone, as in the Intel x86 platform.

point

In typography, a unit of measurement for type (1 point equals approximately 1/72 inch).

Point-to-Point Protocol

An advanced serial packet protocol, similar to but an improvement on SLIP, commonly used for dial-up connections. PPP is an official Internet protocol; SLIP is not.

pointer

- 1** A data type that can contain the address of another data type such as a variable, or of a function or class.
- 2** A graphic image displayed on the screen that indicates the location of a pointing device (sometimes called a cursor).

polygon-filling mode

In graphics, an algorithm that determines the parts of a region that can be filled, painted, inverted, or clipped.

polymorphism

In C++, the concept of a single interface for multiple functions. For example, different classes can each contain a function called `print` that prints data in a format appropriate for objects of that class. The compiler selects the appropriate print function for each call to `print`. See also virtual function.

pop-up menu

A menu that is hidden until the user performs an action (such as clicking the right mouse button) that causes Windows to display the menu. The pop-up menu contains commands that are relevant to the selection or the active window. See also drop-down menu.

pop-up window

An immovable, nonsizable window that remains on the screen until the user dismisses it. Pop-up windows typically contain definitions of terms or other parenthetical information. Good uses for pop-up windows include illustrations, examples, notes, tips, and lists of keyboard shortcuts.

portable code

Code that can be compiled with little or no change to run on more than one platform.

Portable Operating System for Unix

An IEEE standard that defines the language interface between the UNIX operating system and application programs through a minimal set of supported functions.

POSIX

An IEEE standard that defines the language interface between the UNIX operating system and application programs through a minimal set of supported functions.

postfix notation

Placing of operators after operands, as in $xy+z^*$. Compare prefix notation, infix notation.

PPP

An advanced serial packet protocol, similar to but an improvement on SLIP, commonly used for dial-up connections. PPP is an official Internet protocol; SLIP is not.

Practical Extraction and Reporting Language

A general-purpose scripting language used mainly for server-side HTML programming. It is freely available on a wide variety of platforms.

pragma

An instruction to the compiler to perform a given action at compile time. For example, the pragma

#pragma pack([n])

specifies packing alignment for structure and union members.

precompiled header directive

A statement in a source or include file specifying where precompilation should stop.

precompiled header file

A file containing compiled code for a portion of a project. Subsequent builds combine this file with the uncompiled code, thus shortening the overall compile time. The default filename extension for a precompiled header file is .PCH.

predefined macro

A macro recognized by the compiler that takes no arguments and cannot be redefined. In most cases, the value of a predefined macro must be constant throughout compilation. For example, the ANSI predefined macro __TIME__ expands to the most recent compilation time of the current source file, expressed as a string literal of the form *hh:mm:ss*.

predefined query

Or stored procedure. In SQL, a set of one or more SQL statements stored in a data source, available to be called from an application as needed. Predefined queries reduce the overhead of repeatedly specifying the same selection criteria.

prefix character

In Windows text controls, the ampersand (&) character used to define the next character in the string as the mnemonic access key for the control. The ampersand is a directive to underscore the next character in the interface. A two-ampersand (&&) mnemonic-prefix character is a directive to print a single ampersand in the interface.

prefix notation

Placing of operators before operands, as in $*+xyz$. Compare infix notation, postfix notation.

preprocessor

A text processor that manipulates the text of a source file as part of the first phase of translation. The preprocessor does not parse the source text, but it does break it up into tokens for the purpose of locating macro calls. Although the compiler ordinarily invokes the preprocessor in its first pass, the preprocessor can also be invoked separately to process text without compiling.

preprocessor directive

Or compiler directive. An instruction to the preprocessor to perform an action on source-program text before the compilation phase. For example, the directive `#include my_defs.h` instructs the preprocessor to include the contents of the file `my_defs.h` in the compilation unit.

presentation cache

In OLE, a container document's memory cache for the graphical representation of a linked or embedded object. Such a presentation is cached locally in the container document so it can be obtained without the expense of running the object's application. See also presentation data.

presentation data

In OLE, the graphical representation of a linked or embedded object.

primary key

In a database program, a field or group of fields that uniquely identify a record in a table. No two records in a table can have the same primary key value.

primary language

A language, such as English or Japanese, defined for a particular program. In some cases, a sublanguage is also specified. For example, English can be specified as the primary language and Canadian as the sublanguage. See also language identifier (ID).

primary thread

Or main thread. The initial thread of a process. When the primary thread terminates, so does the process. This primary thread is supplied to the operating system by the startup code in the form of a function address. Usually, it is the address of the **main** or **WinMain** function that is supplied.

primary token

An access token that is typically created only by the Windows NT executive layer. The primary token can be assigned to a process to represent the default security information for that process. See also access token, impersonation token, privilege, security identifier (SID).

primary verb

In OLE, the command that is executed when the user double-clicks an OLE item. For most types of OLE items, the primary verb is Edit, which launches the server that created the item. OLE items can support one or more verbs.

print processor

A dynamic-link library that receives information from the print spooler and sends the interpreted information to the graphics engine (another Windows DLL), which converts print processor output into device driver function calls. The printer processes these commands when it prints the image.

private

In C++, describes a limited degree of access to class members, as specified using the private keyword. Access to the specified class member is restricted to member functions and friends of the member or class.

privilege

In Windows NT, a locally unique identifier (LUID) used to control access to an object or service more strictly than is typical with discretionary access control. A system manager uses privileges to control which users are able to manipulate system resources. An application uses privileges when it changes a system-wide resource, such as when it changes the system time or shuts down the system. See also access token, impersonation token, primary token, security identifier (SID).

process

An executing application that consists of a private virtual address space, code, data, and other operating-system resources, such as files, pipes, and synchronization objects that are visible to the process. A process also contains one or more threads that run in the context of the process.

profiler

A development tool for analyzing the run-time behavior of programs. See also function profiling, line profiling.

profiler batch input file

In a profiling operation, a file that provides condensed information to the Visual C++ profiler (PROFILE). The PREP program generates a profiler batch input file the first time the profiler is run on a program. The default filename extension for profiler batch input files is .PBI. See also profiler batch output file, profiler batch text file.

profiler batch output file

An intermediate file generated by the Visual C++ profiler (PROFILE) and used to transfer information between profiling steps. See also profiler, profiler batch input file, profiler batch text file.

profiler batch text file

In a profiling operation, the file generated by the PREP program and used as input to the PLIST program to generate a human-readable profile of the source code. See also profiler, profiler batch input file, profiler batch output file.

program database file

A file used by the build tools to store information about a user's program. The program database file speeds linking during the debugging phase of development by keeping the debugging information separate from the object files.

progress bar control

In MFC, an application window that is used to indicate the progress of a lengthy operation. The control consists of a rectangle that is gradually filled, from left to right, with the system highlight color as an operation progresses.

prolog code

See prolog/epilog code sequence.

prolog/epilog code sequence

In Windows, a compiler-generated code sequence that executes on entry to a function. Prolog code allocates space on the stack, sets up addressing for the variables local to the function, and initializes registers so that variables passed to the function can be conveniently accessed. The epilog code, which executes on exit from the functions, frees this space, restores the original contents of the registers, and ensures that any return value from the function is available to the caller.

property page

A grouping of properties presented as a tabbed page of a property sheet.

property set

In OLE structured storage, information describing a document, stored in a standard format so that other applications can locate and read that information. For example, a document created with a word processor can have a property set describing the author, title, and keywords.

property sheet

A special kind of dialog box that is generally used to modify the attributes of some external object, such as the current selection in a view. A property sheet has three main parts: the containing dialog box, one or more property pages shown one at a time, and a tab at the top of each page that the user clicks to select that page. An example of a property sheet is the Project Settings dialog box in the Microsoft Developer Studio.

protected

In C++, describes a limited degree of access to class members, as specified using the protected keyword. Access to the specified class member is restricted to member functions and friends of the class, as well as member functions of classes derived from the class.

protocol

1 In networking, a formal set of rules governing the format, timing, sequencing, and error control of exchanged messages on a data network; may also include facilities for managing a communications link and/or contention resolution; a protocol may be oriented toward data transfer over an interface, between two logical units directly connected, or on an end-to-end basis between two end users over a large and complex network

2 Rules and conventions of behavior for an action, such as argument passing, or an entity, such as the member functions of a class.

proxy server

Or firewall. A way to protect your local area network from being accessed by others on the Internet. The proxy server acts as a security barrier between your internal network and the Internet, keeping others on the Internet from accessing confidential information on your internal network.

pseudotarget

In a makefile, a label used in place of a filename in a dependency line. The Microsoft Program Maintenance Utility (NMAKE) interprets the label as a file that does not exist and executes the commands in the dependency line. The following is an example of a pseudotarget, `clean`, that instructs NMAKE to remove all of the object files and the executable file from the project directory:

```
clean:
    erase *.obj
    erase *.exe
```


public

In C++, describes a broad degree of access to class members, as specified using the public keyword. The specified class member can be accessed by any function. See also private, protected.

punctuators

In C++, characters (!, %, ^, for example) that have syntactic and semantic meaning to the compiler but do not, of themselves, specify an operation that yields a value. Some punctuators, either alone or in combination, can also be C++ operators or be significant to the preprocessor.

qualified name

In C++, a name used with the binary scope-resolution (::) operator to disambiguate names that are reused within classes, structures, or unions. The name specified after the scope-resolution operator must be a member of the class specified on the left of the operator or a member of its base class(es).

query

A request for records from a data source. For example, a query can be written that requests, essentially, all invoices for Joe Smith, where all records in an invoice table with the customer name Joe Smith would be selected. See also recordset.

queue

A data structure in which elements are added to the end of a list and removed from the head of the list. A priority queue typically removes elements from the list according to some priority value assigned to each element. The system message queue, for example, holds mouse and keyboard input waiting to be processed.

r-value

The value on the right side of an expression. R-values are sometimes used to describe the value of the expression and to distinguish it from an l-value. An r-value can be one or more of the following: variable, constant, literal, mathematical expression, or function. See also l-value.

radio button

Or option button. In graphical user interfaces, a round button operated by the user to toggle an option or choose from a set of related but mutually exclusive options. The radio button has two states, selected (or checked) and cleared (or unchecked). When selected, a black dot appears inside the button's circle. See also radio group.

radio group

A number of radio buttons grouped together in a group-box control. A radio group allows the user to choose from a set of mutually exclusive options; selecting one radio button automatically clears the previously selected button.

radix

- 1** The number base for numeric input and displayed output. Visual C++ allows octal (8), hexadecimal (16), and decimal (10) radixes. The default radix is decimal.
- 2** In SQL, a column that identifies numeric data types; possible values are 10, 2, or NULL.

RAM

Semiconductor-based memory that can be read from and written to by the microprocessor or other hardware devices. The storage locations can be accessed in any order (i.e., randomly). While other types of memory, including read-only memory (ROM), can also be accessed randomly, RAM is generally understood to refer to volatile memory that can be written to as well as read.

random access memory

Semiconductor-based memory that can be read from and written to by the microprocessor or other hardware devices. The storage locations can be accessed in any order (i.e., randomly). While other types of memory, including read-only memory (ROM), can also be accessed randomly, RAM is generally understood to refer to volatile memory that can be written to as well as read.

RAS

A Windows NT Server feature by which a single serial connection provides a remote workstation with host connectivity, NT file services, or Novell file and printing services (NWLink).

raster

A horizontal line of pixels. A raster display device creates images composed of pixels, or dots. Computer monitors and laser printers are examples of raster devices. Devices such as plotters, which create images on paper by drawing lines with a pen, are not raster devices.

raster display

A video monitor that displays an image on the screen from top to bottom as a series of horizontal scan lines. The scan lines are as wide as the smallest visible image on the screen. Within each scan line, individual picture elements, or pixels, can be illuminated. Television screens and most computer monitors are raster displays.

raster font

Or bitmap font, nonscalable font. A font in which each glyph (character or symbol) is of a particular size and style, designed for a specific resolution of device and described as a unique bitmap. See also scalable font, TrueType font, vector font.

raster operation

In computer graphics, a Boolean operation that combines either the values of the pixels in the selected pen and the destination bitmap (binary operation) or the values of the pixels in the source, the selected brush, and the destination (ternary operation). See also raster operation code.

raster operation code

A 32-bit code that defines how Windows combines colors in output operations that involve a current brush, a possible source bitmap, and a destination bitmap. See also raster operation (ROP).

rasterization

The conversion of vector graphics to equivalent images composed of pixel patterns that can be stored and manipulated as sets of bits. See also raster operation (ROP).

raw data

Unprocessed, typically unformatted data. Raw data is a stream of bits that has not been filtered for commands or special characters. More generally, it is information that has been collected but not evaluated.

read-only

Describes information stored in such a way that it can be played back (read) but cannot be changed (written). See also write-only.

record

- 1** (noun) A collection of data about a single entity, such as an account or a customer, stored in a row of a table. A record consists of a group of contiguous columns (sometimes called fields) that contain data of various types. See also recordset.
- 2** (verb) To retain information, usually in a file.

record field exchange

When bulk row fetching is not implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also bulk record field exchange (Bulk RFX), dialog data exchange (DDX), and DAO record field exchange (DFX).

record view

In form-based data-access applications, a form view object whose controls are mapped directly to the field data members of a recordset object and indirectly to the corresponding columns in a query result or table on the data source. See also form view, recordset.

recordset

A set of records selected from a data source. The records can be from a table, a query, or a stored procedure that accesses one or more tables. A recordset can join two or more tables from the same data source, but not from different data sources. See also dynaset, result set, snapshot.

recordset object

In MFC, an instance of class **CRecordset** or class **CDaoRecordset**, or an instance of any class derived from these. See also record field exchange (RFX).

recursive

The ability of a function or a routine to call itself.

red, green, blue

A mixing model, or method of describing colors, in light-based media such as color monitors. RGB mixes percentages of the light-based primary colors (red, green, and blue) to create other colors.

Windows defines these percentages as three 8-bit values called RGB values. Zero percentage of all three primaries, or an RGB value of (0,0,0), produces black and 100 percent of all three primaries, or an RGB value of (255,255,255), produces white. See also red, green, blue, alpha (RGBA).

red, green, blue, alpha

A model that augments the red, green, blue (RGB) method of combining colors in light-based media, with a fourth color component, alpha, which is used to control color blending. In Open GL, an alpha value of 1.0 corresponds to complete opacity and a value of 0.0 corresponds to complete transparency. See also red, green, blue (RGB).

redraw flag

A Boolean parameter to many Windows functions which, when cleared, prevents the content of the given window from being updated (redrawn). When set, this flag allows the window to be repainted with new information.

reentrant

Code written so that it can be shared by several programs (or processes within a single program) at the same time. When code is reentrant, one program or process can safely interrupt the execution of another program or process, execute its own code, and then return control to the first program or process in such a way that the first program or process does not fail or behave in an unexpected way.

reference

- 1** (noun) In C++, a form of pointer that can be used to access a variable by its address. When such a reference is declared, it must be associated with a variable. The reference becomes an alias for that variable. See also address-of (&) operator.
- 2** (verb) To access a variable, often said of elements in an array or in a record.

reference count

A count of the number of pointers that access, or make reference to, an object, allowing for multiple references to a single object. This number is decremented when a reference is removed; when the count reaches zero, the object's space in memory is freed.

referential integrity

In database management, a set of rules that preserves the defined relationships between tables when records are entered or deleted. Enforcing referential integrity would, for example, prevent a record from being added to a related table when there is no associated record in the primary table. See also schema.

region

- 1** In Windows-based applications, a rectangle, polygon, ellipse, or combination of two or more of these shapes used to define a part of the client window to be painted, inverted, filled with output, framed, or used for hit testing.
- 2** More generally, an area dedicated to, or reserved for, a particular purpose.

register

1 (noun) A small (typically 32-bit), named region of high-speed memory located within the microprocessor and used as a holding area for specific, sometimes critical, pieces of data related to activities going on within the system. For example, a register might be used to hold the results of an addition operation or to hold the address of a particular location in the computer's memory. See also register variable.

2 (verb) The process whereby an application or thread provides information about itself in a registration database or with a registration class. For example, an OLE server application registers with the OLE system registration database. See also registry.

register storage class

A declaration for function arguments and local variables requesting that the associated values be held in high-speed memory registers, if these registers are available. This storage class defaults to automatic when register storage is not possible. The Visual C++ compiler does not honor user requests for register variables; instead, it makes its own register choices when global optimizations are on. However, the compiler does honor all other semantics associated with the register keyword when this declaration is used. See also automatic storage class, static storage class.

register variable

In C/C++, a local variable that a program specifies to be stored in a high-speed memory register, if possible, using the keyword register. See also register storage class.

registration entry file

In OLE applications, a text file description of the classes supported by a server application. When a server application is installed in a system, the contents of its registration entry file are merged with the system registry. Registration entry files usually have a .REG filename extension.

registry

Or OLE system registry, Windows NT registry, system registration database. In 32-bit Windows, the database in which configuration information is registered. This database takes the place of most configuration and initialization files for Windows and new Windows-based applications. See also registration entry file.

registry key

A unique identifier assigned to each piece of information in the system registration database.

regular expression

A search string that uses special characters to match a text pattern in a file. For example, an asterisk (*) means zero or more occurrences of the preceding character; thus, `ab*c` matches `ac`, `abc`, `abbc`, `abbbc`, etc.

relational database

A type of database or database management system that stores information in tables—rows and columns of data—and conducts searches by using data in specified columns of one table to find additional data in another table.

relative link

Links that keep the hyperlink intact even if you move the file containing the hyperlink.

relative path

The location of a file, document, or other object as it relates to the current location. The relative path to a target object includes only that portion of the path that is different from the path of the source object. For example, the relative path from c:\animal\mammal\canine\fido to c:\animal\mammal\feline\fluffy is ..\..\feline\fluffy. See also fully qualified path.

relative time

Or time span. The difference between two absolute time values.

release version

A compiled version of a program that does not include the debugging and diagnostic features included in a build compiled in debug mode. For example, a release version will not include the source code contained in **ASSERT** macros. See also debug version.

relocation information

Information that tells the linker what instructions in the object file will need to be patched with addresses of other objects defined in other source files. Those addresses will be known when all the object files are linked together.

Remote Access Service

A Windows NT Server feature by which a single serial connection provides a remote workstation with host connectivity, NT file services, or Novell file and printing services (NWLink).

remote debugging

- 1** To debug an application written for, and running on, a platform other than the one on which you are running Visual C++ (the host machine). The remote machine is connected to the host machine through the serial port or, in the case of a Macintosh, an AppleTalk network connection. See also debug monitor.
- 2** More generally, to debug an application that is running on a computer other than the one on which the debugger resides.

remote machine

- 1 In the case of remote debugging, the machine on which the target application (the one being debugged) is running, as opposed to the host machine, which is running the debugger. See also debug monitor.
- 2 Or remote computer. More generally, any computer that is not in the immediate vicinity, but which is accessible through some type of cable or communications link. See also local machine.

Remote Procedure Call

A widely used standard defined by the Open Software Foundation (OSF) for distributed computing. RPC enables one process to make calls to functions that are part of another process. The other process can be on the same computer or on a different computer on the network.

repeat count

A value specifying the number of times a keystroke is repeated as a result of the user holding down the key.

reserve size

The amount of a resource that is to be allocated (or committed) for a particular use. For example, in the header of a COFF file, the Windows NT–specific field Heap Commit Size specifies the size of the local heap that the linker and loader are to reserve for that file. Only the commit size is initially allocated; the rest is made available one page at a time, until the reserve size is reached.

resource

1 A program block, dialog box template, bitmap, font, or sound, for example, that can be used by more than one program or in more than one place in a program. The use of resources allows alteration of many features in a program without the necessity of recompiling the program from source code. See also resource type.

2 More generally, any part of a computer system or a network that can be allotted to a program or a process while it is running.

resource browser window

A window that displays the resources associated with a project. From the resource browser window, an application's resources can be added, deleted, or changed.

resource compiler

An application that creates a binary resource file based on the resource-definition (.RC) file. The resource compiler can also append binary resource data to an executable file and create a resource table in the executable file's header.

resource editor

A specialized utility program that can be used to add, edit and delete program resources.

resource fork

The portion of an Apple Macintosh file containing reusable items of information that the program can use during execution, such as blocks of code, icons, fonts, and digitized sounds. See also data fork.

resource identifier (ID)

In Microsoft Windows and in the Apple Macintosh Operating System, a number that identifies a particular resource within a given resource type—for example, a particular menu among many resources of type MENU that a program can use. The resource type, along with the resource ID, uniquely defines every resource in an application. (Note that resources of different types can have the same resource ID.)

resource type

A category of structural or procedural resources in Microsoft Windows and in the Apple Macintosh Operating System, such as code, fonts, dialog boxes, or icons. The resource type, along with the resource ID, uniquely defines every resource in an application.

resource-definition file

Or resource script file. A text file containing descriptions of resources from which the resource compiler creates a binary resource file. For Microsoft Windows applications, resource-definition files usually have a .RC filename extension. For Apple Macintosh applications, such files are typically named with a .R extension and written with the Apple Rez script language. See also compiled resource file.

RESOURCE.H file

The default name for the header file that corresponds to an application's resource-definition file.

response file

A text file that can contain multiple options and filenames that would otherwise be typed on the command line or specified by using the CL environment variable. The options in the response file are interpreted as if they were present on the command line at the position of the response file invocation.

restored window

A window that has been returned to its preminimized or premaximized size and position.

result set

A collection of data returned by an SQL query on a database. A Known result set is when the database application knows the exact form of the SQL statements and, therefore, the form of the result set, at compile time. An Unknown result set is when the application does not know the exact form of the SQL statement at compile time and, therefore, cannot predict the format of these results prior to execution. See also recordset.

return code

Or result code, exit code. A predefined code used to report the outcome of a procedure or to influence subsequent events when a routine or process terminates (returns) and passes control of the system to another routine. The return code can indicate success or failure, or can even indicate a particular type of failure that is within the normal range of expectations. For example, the exception-handling functions in C++ generate return codes.

return type

The data type returned by a function. The type-specifier field of a function definition can specify any fundamental, class, structure, or union type, or **void**.

return value

The value that a function evaluates to in its calling expression.

Rez

A script language that is part of the Macintosh Programmer's Workshop and is used to specify the resources to be used by a Macintosh applicaton. MRC.EXE, the Macintosh Resource Compiler, provides a Windows NT implementation of Rez.

RFX

When bulk row fetching is not implemented, the mechanism by which MFC ODBC classes transfer data between the field data members of a recordset object and the corresponding columns of an external data source. See also bulk record field exchange (Bulk RFX), dialog data exchange (DDX), and DAO record field exchange (DFX).

RGB

A mixing model, or method of describing colors, in light-based media such as color monitors. RGB mixes percentages of the light-based primary colors (red, green, and blue) to create other colors.

Windows defines these percentages as three 8-bit values called RGB values. Zero percentage of all three primaries, or an RGB value of (0,0,0), produces black and 100 percent of all three primaries, or an RGB value of (255,255,255), produces white. See also red, green, blue, alpha (RGBA).

RGBA

A model that augments the red, green, blue (RGB) method of combining colors in light-based media, with a fourth color component, alpha, which is used to control color blending. In Open GL, an alpha value of 1.0 corresponds to complete opacity and a value of 0.0 corresponds to complete transparency. See also red, green, blue (RGB).

RGBA mode

In OpenGL, a color pixel data mode. An OpenGL context is in RGBA mode if its color buffers store red, green, blue, and alpha color components instead of color indexes.

rich-text format file

A file that contains encoded, formatted text and graphics for easy transfer between applications. The rich-text encoding format is commonly used by document-processing programs such as Microsoft Word for Windows and for generating online Help files. Rich-text format files usually have a .RTF filename extension.

right outer join

A database term that describes the relationship between two tables on which a query is run. If two tables are connected with a right outer join, the resulting recordset contains all records from the table on the right but only those records from the table on the left where the joined fields are equal. See also left outer join.

risk management

The total process of identifying, controlling, and eliminating or minimizing the effects of uncertain events, such as power failures or hacker activity, that may affect system resources. It includes risk analysis, cost benefit analysis, selection, implementation and test, security evaluation of safeguards, and overall security review.

RLE8 compression

An image compression format for device-independent bitmaps that encodes a sequence of identical pixels as a repeat count and a color value.

rollback

A database operation that allows recovery from changes made during a transaction that was canceled or failed. Rollback returns the records to the state they were in as of the last commit (saved transaction).

root

In a hierarchy of items, the one item from which all other items are descended. The root item has nothing above it in the hierarchy. See also inheritance hierarchy.

ROP

In computer graphics, a Boolean operation that combines either the values of the pixels in the selected pen and the destination bitmap (binary operation) or the values of the pixels in the source, the selected brush, and the destination (ternary operation). See also raster operation code.

rotation

A transformation that makes a displayed graphical object appear to have been turned (rotated) in space, relative to the coordinate-space origin.

router

A hardware device that connects two or more networks, or connects a network to the Internet. It converts addresses and sends on only those messages that need to pass to the other network.

rowset

In ODBC, one or more rows returned by a single fetch operation.

RPC

A widely used standard defined by the Open Software Foundation (OSF) for distributed computing. RPC enables one process to make calls to functions that are part of another process. The other process can be on the same computer or on a different computer on the network.

RTTI

In C++, a mechanism that allows the type of an object to be determined during program execution.

rubber-band selection

A selection method that allows a user to select multiple items by dragging a sizing rectangle around the items to be selected. Items within the region can be manipulated by the user. For instance, the user might drag or drop the selection into another container application.

run time

The point in time when a program is running, or the period of time needed to execute the program.
See also compile time, link time.

run-time class information

Information about an object's class accessed at run time. This information is useful for doing extra type-checking or creating dynamic objects at run time. Classes derived from the MFC class **CObject** can have this functionality. See also run-time type information (RTTI).

run-time error

A math or logic error in a program that becomes apparent only at run time. Such an error may produce incorrect results or cause the program to behave unpredictably or crash.

run-time type information

In C++, a mechanism that allows the type of an object to be determined during program execution.

running state

In OLE, the status of a compound document object (either a linked or embedded item) when the object's application is running and it is possible to edit the object, access its interfaces, and receive notification of changes. See also loaded state.

sampling frequency

Or sampling rate. The rate at which samples of a physical variable, such as sound, are taken. The more samples taken per unit, the more closely the reconstructed result resembles the original.

scalable font

Any font that is defined by mathematical routines that can reproduce the outlines of each character at any size. Macintosh System 7, vector, PostScript, and TrueType fonts are all scalable; raster (bitmap) fonts are not.

scalar type

Or scalar data type. A data type that has a predictable and enumerable sequence of values that can be compared for greater-than/less-than relationships. In Visual C++, all arithmetic types, plus pointers, are considered scalar types. See also aggregate type.

scaled index base indirect operand

In Intel 80386-80486 assembly language, the second encoding byte of an extended memory operand.

scaling

A transformation that alters the apparent size of an object.

scan code

A device-dependent identifier that uniquely identifies each key on a keyboard. A keyboard generates two scan codes when the user types a key—one when the user presses the key and another when the user releases the key. See also virtual key code.

scan line

- 1** On a television or raster-scan computer monitor, one of the horizontal lines on the inner surface of the screen that is traced by the electron beam to form an image.
- 2** A row of pixels read by any scanning device, such as a full-page scanner or a fax machine.

schema

A description of the current structure of tables and views in a data source. The schema describes what columns are in each table, the data type of each column, and the relationships between tables. See also referential integrity.

schema number

In MFC serialization, the version of a class implementation, assigned to a class when the **IMPLEMENT_SERIAL** macro of the class is encountered. The schema number refers to the implementation of the class, not to the number of times a given object has been made persistent (usually referred to as the object version). Do not confuse this schema number with database terminology.

scope

In programming, the extent to which a given identifier (constant, variable, data type, routine) can be referenced within a program. See also class scope, file scope, function scope, function-prototype scope, local scope.

scope-resolution (::) operator

In C++, the operator with highest precedence, used to define the scope of the operand. The unary form of the operator, `::foo()`, is used to uncover or access a name that is at global scope and has been hidden by local or class scope. The binary form, `Bar::foo()`, is used to disambiguate names that are reused within classes, structures, or unions.

screen coordinates

A means of specifying the position of a point on the display screen in terms of vertical (y-coordinate) and horizontal (x-coordinate) displacement from the upper-left corner of the screen (origin). The position and size of a window can be described by one set of coordinates (x, y) that marks the point defining the upper-left corner of the window, and another set of coordinates (x', y') that marks the point defining the lower-right corner of the window.

script file

A type of program that consists of a set of instructions to an application or utility program. The instructions in a script are usually expressed using the application's or utility's rules and syntax, combined with simple control structures such as loops and **if-then** expressions. A resource-definition file and a batch (.BAT) file are two examples of script files.

scroll-bar code

A 2-byte value that indicates the user's scrolling request. For example, the constant SB_LEFT indicates a request to scroll to the far left and the constant SB_PAGELEFT indicates a request to scroll one page to the left.

scroll-bar control

A control window that belongs to the SCROLLBAR window class. A scroll-bar control appears and functions like a standard scroll bar, but it is a separate window that receives direct input focus, indicated by a flashing caret displayed in the scroll box. Unlike a standard scroll bar, a scroll-bar control also has a built-in keyboard interface that enables the user to direct scrolling. See also standard scroll bar.

scrolling

The process of moving a document in a window to permit viewing of any desired portion.

scrolling range

The minimum and maximum values that a scroll bar can report.

SDI

A user interface architecture that allows a user to work with just one document at a time. Windows Notepad is an example of an SDI application. See also multiple document interface (MDI).

SDK

A set of libraries, header files, tools, books, on-line help and sample programs designed to help a developer create software.

sector

On a disk, the smallest contiguous physical unit for recording information. Multiple sectors make up a track.

Secure Sockets Layer

A protocol for providing data security layered between its service protocols (HTTP) and TCP/IP.

security attribute

A characteristic of a file or object that regulates its access and privileges by users or other objects.
See also access token.

security descriptor

Contains the security information associated with an object. The information in security descriptors can include an owner, a primary group, a discretionary access-control list, and a system access-control list. This information is stored in the form of security identifiers (SIDs) and access-control lists (ACLs).

security identifier

A structure of variable length that uniquely identifies a user or group on all Windows NT implementations. See also access token, impersonation token, primary token, privilege.

security policy

The set of laws, rules, and practices that regulates how an organization manages, protects, and distributes sensitive information.

SEH

A mechanism for handling hardware- and software-generated exceptions that gives developers complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and computers. See also C++ exception handling.

selection statement

A statement that provides a means to conditionally execute sections of code. The **if** and **switch** statements are C/C++ selection statements.

semantics

The relationships between words or symbols and their intended meanings, or the rules governing these relationships. See also syntax.

semaphore

1 In Win32, a synchronization object that maintains a count between zero and a specified maximum value. A semaphore's state is signaled when its count is greater than zero and nonsignaled when its count is zero. The semaphore object is useful in controlling a shared resource that can support a limited number of users. It acts like a gate that counts the threads as they enter and exit a controlled area and that limits the number of threads sharing the resource to a specified maximum number.

2 More generally, a flag variable used to govern concurrent processes that share system resources.

separator

- 1** In Windows, a special type of menu item that appears as a horizontal line. A separator can be used in a pop-up menu to divide a menu into groups of related items.
- 2** Or separator code. A Unicode value used to indicate line breaks (0x2028) or paragraph breaks (0x2029).
- 3** More generally, a keyword, character, or white space used to separate components of a name, a number, or a group of fields. For example, the decimal point (.) in 123.456 and the backslash (\) in ROOT\SUBDIR are separators.

Serial Line Internet Protocol

A protocol for connecting to the Internet via a dial-up connection, such as with a modem.

serial port

An electrical connection to a computer through which data is transmitted in series, one bit after another.

serial transport layer

A data transport link established by connecting the serial ports of two machines. For example, a serial transport layer is established when an Apple Macintosh and a Win32 host are connected through their serial ports.

serialization

Or object persistence. In MFC, the process of writing or reading an object to or from a persistent storage medium, such as a disk file. The basic idea of serialization is that an object should be able to write its current state, usually indicated by the value of its member variables, to persistent storage. Later, the object can be re-created by reading, or deserializing, the object's state from storage.

serif

Any of the short lines or ornaments at the ends of the strokes that form a character in a typeface.

server

- 1** In a network, any device that can be shared by all users.
- 2** An application or a process that responds to a client request. See also client/server.

server application

An application that can create OLE items for use by container applications. Data in a server application can usually be copied, using the Clipboard or a drag-and-drop procedure, so that a container application can paste the data as an embedded or linked item. An application can be both a container and a server. See also container application, mini-server application.

server document

In OLE, a document created by a server application. See also compound document.

server item

An object that provides an interface between an OLE item and the server application and that is of a class derived from the MFC class **COleServerItem**. Server items, which are created and maintained by the server application, can be either linked or embedded. For linked items, the server item provides access to the data source, often in a different file. For embedded items, the server item handles the data stored in the container document and creates the server document. Server items also generate presentation data and handle the verbs (commands) associated with the OLE item.

service

In Win32, an executable object that is installed in a registry database maintained by the Service Control Manager. The executable file associated with a service can be started at boot time by a boot program or by the system, or it can be started on demand by the Service Control Manager. The two types of service are Win32 service and driver service.

session

- 1** In a Windows NT client/server network, a link between a workstation and a server. The session is established the first time a workstation makes a connection with a shared resource on the server and ends when all current connections between the workstation and the server are deleted.
- 2** In the ISO/OSI communications model, the protocol layer that provides a way for users to establish connections and transport data across those connections. See also transport layer.
- 3** More generally, the time during which two computers (or a computer and a terminal) maintain a connection, or the connection itself.

SGML

A set of rules and tags to mark the structure and content of a document, independent of the display medium.

shared library

In general, any code module that can be accessed and used by many programs. Shared libraries are used primarily for sharing common code between different executable files or for breaking an application into separate components, thus allowing easy upgrades. In the Visual C++ documentation, shared library usually refers to a code module that is an Apple Shared Library Manager (ASLM) file for the Apple Macintosh. In Windows, shared libraries are usually referred to as dynamic-link libraries (DLLs).

shared memory

Memory accessed by more than one process or thread in a multitasking environment. Processes or threads using this memory operate under a set of rules that prevent them from modifying the same addresses simultaneously.

shared resource

- 1 In a Windows NT client/server network, any local resource on the server that is made available to network users, such as directories, files, printers, or named pipes.
- 2 More generally, any device, data, or program that is used by more than one device, program, process, or thread.

sharing mode

A file opening mode that determines what, if any, read and write operations will be allowed when the file is being shared. See also shared resource, sharing violation.

sharing violation

An error that occurs when one process (or machine) attempts to access a file after a different process has requested that the server block access to the file. If an application opens the file in compatibility mode, a sharing violation results in a critical error. See also sharing mode.

shell

A piece of software, usually a separate program, that provides communication between the user and the operating system. For example, the Windows Program Manager is a shell program that interacts with MS-DOS.

SHIFT+F1 Help

Context-sensitive Windows Help that the user obtains by pressing the SHIFT and F1 keys together. SHIFT + F1 Help invokes a special Help mode in which the cursor turns into a Help cursor. The user can then select a visible object in the user interface, such as a menu item, toolbar button, or window. This opens Help on a topic that describes the selected item.

short integer

In 32-bit Visual C++, a 16-bit integer. Most compilers provide less storage space for a **short** than for an **int**; however, the ANSI standard guarantees only that a short integer is no longer than an integer, which is the machine's native (word) size.

shortcut

A fast way to perform an action such as selecting text or, more usually, opening a file, document, Web page, and so on. Usually represented by an icon on the desktop.

shortcut key

See accelerator key.

shortcut menu

A menu displayed within a window that provides quick access to frequently used commands that are also available from the main menu bar. The commands in a shortcut menu may change depending on the current state of the window.

show state

A collection of qualities that a windows has at a given time, including active or inactive; hidden or visible; and minimized, maximized, or restored.

SIB

In Intel 80386-80486 assembly language, the second encoding byte of an extended memory operand.

sibling

A node in a tree that is descended from the same immediate ancestor(s) as other processes or nodes. The node may represent any data structure, or a system object such as a window or a process.

sibling window

A child window that has the same parent window as one or more other child windows.

SID

A structure of variable length that uniquely identifies a user or group on all Windows NT implementations. See also access token, impersonation token, primary token, privilege.

side effect

A change of state, other than the obvious one, caused by a routine, function call, or assignment. Function calls can have side effects if they change the value of an externally visible item.

signature

- 1** In Win32, a 4-byte value that identifies an enhanced metafile.
- 2** More generally, a sequence of data used for identification, such as an identifier appended to an electronic mail message or in a fax.

signed integer

An integer data type that can be either positive or negative. The most significant bit is the sign bit, which is 1 for negative values and 0 for positive values. See also unsigned integer.

signed values

Values that can be negative or positive.

Simple Mail Transfer Protocol

A standard Internet protocol for sending e-mail documents.

Simple Network Management Protocol

Designed for requesting, packaging, and sending management information over a network. SNMP provides a common set of rules for programmers writing network management programs.

single document

A user interface architecture that allows a user to work with just one document at a time. Windows Notepad is an example of an SDI application. See also multiple document interface (MDI).

single-byte character set

A mapping of characters to their identifying numeric values, in which each value is 1 byte wide. The ANSI and OEM character sets are single-byte character sets. See also multibyte character set (MBCS), Unicode.

single-line edit control

An element of the Windows user interface that allows the user to enter and edit a single line of text.
See also multiline edit control.

size box

A small rectangle the user can manipulate to change the size of a window.

sizing border

A type of window border that enables the user to size the window by clicking and dragging the border.

sizing handles

A mechanism for changing the size of a bitmap or a control. Active sizing handles are solid squares; if a sizing handle is a hollow square, the object cannot be resized along that axis.

skeleton application

Or starter application. A default application created by AppWizard that runs, opens and closes windows, and allows other operations on the windows. You add the necessary code to implement the functionality needed for your own application.

slider control

A window containing a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the control sends notification messages to indicate the change.

SLIP

A protocol for connecting to the Internet via a dial-up connection, such as with a modem.

small memory model

A memory model with only one code segment and only one data segment.

smart pointer

In C++, an object that implements the functionality of a pointer and additionally performs some action whenever an object is accessed through it. Smart pointers are implemented by overloading the pointer-dereference (->) operator.

SMTP

A standard Internet protocol for sending e-mail documents.

snapshot

- 1** In MFC, a recordset that reflects a static view of the data as it existed at the time the snapshot was created. See also dynaset, recordset.
- 2** Or screen dump. A copy of all or part of the display screen as it appears at a given instant.
- 3** More generally, a copy of an object's state or appearance at a given time.

SNMP

Designed for requesting, packaging, and sending management information over a network. SNMP provides a common set of rules for programmers writing network management programs.

socket

An object that represents an endpoint for communication between processes across a network transport (TCP/IP or AppleTalk, for example). Sockets have a type (datagram or stream) and can be bound to a specific network address. Windows Sockets provides an API for handling all types of socket connections in Windows. See also datagram socket, stream socket, transport protocol.

software development kit

A set of libraries, header files, tools, books, on-line help and sample programs designed to help a developer create software.

solid brush

A logical brush that contains 64 pixels of the same color. See also null brush.

sort order

The order in which a set of records or other data objects are to be sorted, or the function that defines this order. Possible sort orders for an array of strings, for example, could include lexicographic order or ascending order by length.

source character set

The set of legal characters that can appear in source files. For Microsoft C and C++, the source set is the standard ASCII character set. The source character set and execution character set include the ASCII characters used as escape sequences. See also execution character set.

source code

Human-readable statements written in a high-level programming language, or assembly language.
See also object code.

source code editor

A text editor that may provide special formatting features which make it easier to generate readable, syntactically correct source code. For example, a source code editor may automatically indent blocks of code, check for balanced parentheses and brackets, or highlight keywords.

spawn

- 1 (noun) A family of C run-time library functions that create and execute new child processes.
- 2 (verb) To create a child process (or thread, in cases where multiple threads are allowed).

spline

In computer graphics, a curve calculated by a mathematical function that connects separate points with a high degree of smoothness.

SQL

A database sublanguage used to query, update, and manage relational databases.

SSL

A protocol for providing data security layered between its service protocols (HTTP) and TCP/IP.

stack

- 1 A data structure implemented as a LIFO (last in, first out) list so that the last item added to the structure is the first item removed.
- 2 A region of reserved memory, organized as a stack, in which programs temporarily store status data such as procedure and function call return addresses, passed parameters, and local variables. See also heap.

stack allocation

The amount, in bytes, of space reserved for a program's status data such as procedure and function call return addresses, passed parameters, and local variables. See also stack frame.

stack frame

Or frame allocation. An area of memory set up whenever a function is called that temporarily holds the arguments to the function as well as any variables that are defined local to the function. There are two key characteristics of frame allocations. First, when a local variable is defined, enough space is allocated on the stack frame to hold the entire variable, even if it is a large array or data structure. Second, frame variables are automatically deleted when they go out of scope.

stack overflow

An error condition caused by attempting to push an item onto a stack that is full, meaning that all of the memory allocated for that stack has been used. See also stack underflow.

stack probe

A short routine, called on entry to a function, to verify that there is enough room in the program stack to allocate local variables required by the function.

stack size

The amount of memory, in bytes, allocated to a stack.

stack underflow

An error condition caused by attempting to pop an item from an empty stack. See also stack overflow.

stand-alone code

For the Apple Macintosh and Power Mac, code that implements resources as a shared library, with some of the functionality of dynamic-link libraries. See also Apple Shared Library Manager (ASLM).

standard control

One of the controls provided by Microsoft Windows. These controls include buttons of several kinds, static- and editable-text controls, scroll bars, list boxes, and combo boxes. See also custom control.

standard conversion

In C++, the conversion of objects of one fundamental type to another type. For example, converting an object of integral type to a shorter signed or unsigned integral type. Standard conversion can result in loss of data if the value of the original object is outside the range that can be represented by the shorter type.

standard error device

The device to which a program sends its error messages unless the error output is redirected. Normally, the standard error device is the console.

Standard Generalized Mark-up Language

A set of rules and tags to mark the structure and content of a document, independent of the display medium.

standard input device

The device from which a program reads its input unless the input is redirected. In normal operation, the standard input device is the keyboard.

standard input/output (I/O)

In C, the input and output functions declared in the `STDIO.H` header file. See also standard input device, standard output device, standard error device.

standard output **device**

The device to which a program sends its output unless the output is redirected. In normal operation, the standard output device is the console.

standard resource

A resource whose format is defined and recognized by Windows. Standard resources include icons, cursors, menus, dialog boxes, bitmaps, fonts, keyboard accelerator tables, message-table entries, string-table entries, and version data. See also custom resource.

standard scroll bar

One of two ways to include a scroll bar in a window. A standard scroll bar is located in the nonclient area of a window. It is created with the window and displayed when the window is displayed. The sole purpose of a standard scroll bar is to enable the user to generate scrolling requests for viewing the entire content of the client area.

start page

A page the user chooses as the opening page of the Internet or a Web site.

starter files

In Visual C++, a set of files created by AppWizard that, when compiled, implement the basic features of a Windows application. The starter files consist of C++ source files, resource files, header files, and a project file. See also skeleton application.

startup code

The portion of the program code that gets an application up and running. Startup code interprets command-line arguments, creates and initializes global variables, opens standard streams, and so forth.

state flags

Values, often return values from functions, that specify the condition of an interface component such as a checkbox.

static control

A control that enables an application to provide the user with certain types of text and graphics that require no response. Applications often use static controls to label other controls or to separate a group of controls.

static cursor

An ODBC cursor that appears to be fixed (static) from the perspective of the data in the underlying tables. Changes made by other users are not detected by the static cursor until it is closed and reopened. See also snapshot.

static data

Data declared with the keyword **static**. Static data can include initialized variables defined outside of functions, static variables within functions, explicit strings, and floating-point numbers. Static data also sometimes refers to the area in memory where static data resides. See also statically allocated buffer.

static data member

Or static member variable. A data member that is declared within the scope of a class but which is actually a separate object. The definition of the static data member is performed elsewhere in the program, and only one copy of the member exists, no matter how many objects of the class exist.

static extent

A property of global data objects (both **static** and **extern**), local static objects, and static data members of C++ classes that have the following characteristics:

- Only one copy of the data is maintained for all objects.
- The objects retain their location in memory from the time they are created until they are destroyed.

static link

A program link, to a library or to an object, that is established at link time. When an application uses a function from a static-link library, the linker copies the code for that function into the application's executable file. See also dynamic link.

static splitter window

A split-window style in which the panes are created when the window is created, and the order and number of panes never change. The panes are separated by a splitter bar that the user can drag to change the relative sizes of the panes. See also dynamic splitter window.

static storage class

In C++, the storage class for objects and variables that exist and retain their values throughout the execution of the entire program. All global objects have static storage class. Local objects and class members can be given static storage class by explicit use of the **static** storage class specifier. See also automatic storage class.

static-link library

A library file that is linked into the program when the executable file is built. Static-link library files usually have a .LIB filename extension. See also dynamic-link library file, library file.

statically allocated buffer

Or static buffer. A portion of memory that is allocated when a module is loaded and is deallocated when the module leaves memory.

status bar

A control bar at the bottom of a window, with a row of text output panes. The status bar is usually used as a message line (for example, the standard menu help message line) or as a status indicator (for example, the CAP, NUM and SCRL indicators). See also dialog bar.

status code

A value used to report on the current status of an object, event, or process, or to reflect the outcome of an operation.

storage class

Or storage duration. In C/C++, determines whether a variable (or object, in C++) has a static (or global) lifetime, in which case it is stored in the same memory location throughout the execution of the program, or an automatic (or local) lifetime, (in which case it is allocated new storage each time execution control passes to the block in which it is defined. See also automatic storage class, register storage class, scope, static storage class.

storage object

An object type used in OLE to implement compound files. Storage objects are analogous to directories in that they can contain other storage objects, or they can contain stream objects, which are analogous to files. See also compound file.

stream file object

A virtual file representing on-disk data associated with a file, some of which may not be part of the physical file that backs a file object. For example, a stream file object makes it possible to cache the extended attributes (EAs) or access-control list (ACL) for a file object together with the file's data.

stream I/O

Or iostream. In C++, the input and output functions, declared in IOSTREAM.H, that transfer data from and to files and devices. Stream I/O functions treat data as a stream of individual characters and provide buffering. The predefined object **cout** represents the standard output stream, **cin** represents the standard input stream, and **cerr** represents the standard error stream.

stream object

One of the object types used in OLE to implement compound files. Stream objects store data of any type. See also compound file, storage object.

stream socket

A connection-oriented socket that provides a bidirectional, sequenced, and unduplicated flow of data without record boundaries. Receipt of stream messages is guaranteed, and streams are well-suited to handling large amounts of data. Stream sockets are appropriate, for example, for implementations such as file transfer protocol (FTP), which facilitates transferring ASCII or binary files of arbitrary size. See also datagram socket, transport protocol.

streaming

The process of transferring information from a storage device, such as a hard disk or CD-ROM, to a device driver. Rather than transferring all the information in a single data copy, the information is transferred in smaller parts over a period of time, typically while the application is performing other tasks.

strikeout

A font effect that adds a horizontal line through one or more characters.

string

A data structure composed of a sequence of characters identified with a symbolic name. In C/C++, a string is terminated with a null character ('\0').

string constant

In C/C++/Java programming, a list of characters enclosed in double quotes in source code.

string identifier (ID)

A 16-bit identifier that Windows uses to locate a string in a string-table resource. The upper 12 bits of the identifier specify the block in which the string appears. The lower 4 bits specify the ordinal location of the string within the block. See also string table.

string literal

Or literal string, string constant. A string of characters enclosed with double quotation marks ("). Any character from the source character set is allowed, except that a double quotation mark inside the string must be preceded by the backslash, or escape, character (\). Like other constants, string literals do not change in a program. See also character constant.

string name

An identifier for a class, version, or resource, in the form of a human-readable character string.

string resources

In 32-bit Windows, null-terminated Unicode strings that are stored in the resource file string table. Each string is made up of a series of strings whose ordinal position is used as the string ID.

string table

1 A Windows resource that contains a list of identifiers, values, and captions for the strings used in an application's framework. For example, the status bar prompts are located in the string table.

When the resource compiler converts a string table specified in a resource-definition file, it separates it into blocks of 16 strings and stores them as individual resources. See also string resources.

2 More generally, any data structure used to store character strings. Typically, a string table is implemented as a hash table.

structure

1 In C, an aggregate data type that can contain constants, variables, and other structures. In C++, a structure can also contain functions and all of a structure's members are implicitly public.

2 More generally, a collection of data elements.

structured exception

A mechanism for handling hardware- and software-generated exceptions that gives developers complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and computers. See also C++ exception handling.

Structured Query Language

A database sublanguage used to query, update, and manage relational databases.

structured storage

An OLE model that allows objects to control their own data storage, loading directly from and saving directly to disk.

stub

An interface-specific object that unpackages the parameters for that interface after they are marshaled across the process boundary, and makes the requested method call. The stub runs in the address space of the receiver and communicates with a corresponding proxy in the sender's address space.

stub file

Or stub program. In Windows, an MS-DOS executable file added to the beginning of a segmented executable file and invoked if a user tries to run a Windows program from the MS-DOS prompt. The stub may display some error message such as This program requires Microsoft Windows.

style

A value, or set of values, that defines the outward appearance and behavior of an object, such as a window, control, or document. See also window style.

style bit

An individual bit of the 16-bit style parameter that pertains to a single style attribute. For example, the `WS_VISIBLE` style bit, when set, determines whether a particular window is visible to the user.

subaddress

The part of a URL that goes to a specific place in a file, such as a bookmark, slide, and so on.

subclass

Or derived class. The class that is derived from another class. A subclass inherits state and behavior from its superclass or superclasses in the form of variables and methods.

subclassing

In Windows programming, a technique that allows an application to intercept and process messages sent or posted to a particular window before the window has a chance to process them. By subclassing a window, an application can augment, modify, or monitor the behavior of the window.

subexpression

An expression that is part of a larger expression. For example, $(a+b)$ is a subexpression of $(a+b)^*c$.

sublanguage

Or secondary language. In the localization of Windows-based programs, a variant of the primary language, defined by the locale. For example, if English is the the primary language, American, British, Australian, New Zealand, Canadian, and Ireland are the possible locales that determine the sublanguage. See also language identifier (ID).

subobject

- 1 The portion of a Windows object that is completely described by a base class.
- 2 Generally, an object within an object. For example, a cell could be considered a subobject of a spreadsheet object.

subscript

- 1 In programming, a value enclosed in brackets ([]) that indicates either the number of elements in an array in the array declaration or the offset position within an array.
- 2 In printing, one or more characters printed slightly below the bottom edge of the surrounding text.

subscript ([]) operator

An operator that indicates that the name preceding the operator is an array, or that designates a subscript into the array. For classes that have overloaded the operator, the behavior of this operator is class-specific.

substring

A string that is part of a longer string. For example, the string cat is a substring of the string catamaran.

subsystem

A system other than, and usually subordinate to, the primary system. A subsystem can have its own memory allocation and internal functions. OLE 2.0 and DEBUG are examples of subsystems, and POSIX is a subsystem of Windows NT that can run UNIX applications.

superclass

Or base class. A class which is the base (or parent) class for another class; a class from which another is derived, either directly or indirectly. The superclass provides state and behavior to the subclass which the subclass may modify by overriding selected methods.

suspend count

A record of the number of active operations that require a thread to momentarily suspend execution of user-mode code. Starting one of these operations increments the suspend count; ending the operation decrements the count. When the suspend count equals zero, the thread resumes execution.

swap file

Or paging file. In Windows, the disk file that holds the active system and application memory pages that are not currently present in main memory (RAM). See also virtual memory.

symbol

- 1** A character other than the standard alphanumeric characters. It usually refers to algebraic, scientific, or linguistic characters not found on the keyboard.
- 2** In programming, a name that represents a register, an absolute value, or a memory address (relative or absolute).
- 3** To a compiler, a variable, function name, or other identifier.
- 4** In Visual C++, a resource identifier that consists of a text string (name) mapped to an integer value. A symbol provides a way to refer to resources and user-interface objects, both in source code and in the resource editors.

symbolic-debugging information

A map of the source code and all the identifiers (variables, function names, and so on) created at compile time for use by the debugger. See also program database (.PDB) file.

synchronization object

An object whose handle can be specified in one of the wait functions to coordinate the execution of multiple threads. The state of a synchronization object is either signaled, which can allow the wait function to return, or nonsignaled, which can prevent the function from returning. More than one process can have a handle of the same synchronization object, making interprocess synchronization possible. See also mutex object, semaphore.

synchronous operation

- 1** In Windows programming, a task that requires the thread that initiated the operation to suspend activity until the task is completed. See also asynchronous operation, atomic operation.
- 2** In hardware, an operation that proceeds under control of a clock or timing mechanism.

synchronous processing

In ODBC, a method of processing transactions in which the database driver does not return control to an application until a function call completes. See also asynchronous processing.

syntax

The grammar of a particular language, the rules governing the structure and content of the statements. See also semantics.

System 7

An operating system for Apple Macintosh computers.

system database

A file, read at startup, that contains information about the users in a workgroup, such as account names, user preference information, and passwords.

system font

The font used by the operating system to display messages. The system font is the default font for resources.

system modal dialog box

Or system modal message box, system modal window. A dialog box that prevents the user from doing anything else in Windows until the dialog box is cleared, usually by choosing a pushbutton marked either OK or Cancel. Use a system modal dialog box to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory).

system palette

A representation of the device's physical palette. The system palette contains the RGB values for all colors that can currently be displayed or drawn by the device.

system time

The current time on the system's real-time clock. The system time structure contains values for the year, month, day, hour, minute, second, and millisecond.

SYSTEM.INI file

A Windows initialization file that contains the settings needed to configure Windows to a system's particular components.

tab order

The order in which the `TAB` key moves the input focus from one control to the next within a dialog box. Usually, the tab order proceeds from left to right in a dialog box, and from top to bottom in a radio group.

tab stop

One of the points in a line of text or a control in a group of controls (in a dialog box, for example) that the user can move to by pressing the TAB key. See also tab order.

tab-delimited report

A data file in which the elements are separated by tab characters.

tag

- 1** In C/C++, an optional identifier are part of structure, union, and enumeration type specifiers and, if present, always immediately follow the reserved words **struct**, union, or **enum**. The tag names must be distinct from all other structure, enumeration, or union tags with the same visibility.
- 2** Text in angle brackets that represents HTML markup. Web browsers display text and graphic elements based on the tags an author uses. The tag itself is not displayed by the browser.

tail

The last element in a linked list.

TAPI

A set of functions that is part of the Win32 API that lets a computer communicate directly with telephone systems.

target

The objective, or destination, of a computer command or operation. For example, the target machine in a remote debugging operation is the machine running the application that is being debugged.

task handle

One of two handles that Windows creates for each task running in the system. The task handle is the handle to the task database (TDB), which contains information about the task's queue, module handle, and so forth. See also instance handle.

TCP/IP

A set of transport protocols for the Internet that provides both connection-oriented (TCP) and connectionless (IP) data transfer. Commonly made up of four protocols: IP, TCP, UDP, and ICMP. See also transport protocol, User Datagram Protocol (UDP).

Telephony Application Programming Interface

A set of functions that is part of the Win32 API that lets a computer communicate directly with telephone systems.

teletype network

Terminal-emulation protocol for remote login over the Internet. Also refers to a UNIX program that uses the protocol (often written TELNET in that case).

telnet

Terminal-emulation protocol for remote login over the Internet. Also refers to a UNIX program that uses the protocol (often written TELNET in that case).

template

- 1** In C++, a keyword that allows polymorphism with respect to different types, by passing the data type as a parameter to the code body.
- 2** More generally, a form or blueprint for an object that contains information about the default properties of that object. For example, a Microsoft Word document template may contain text, formatting, and graphics information as well as macros and AutoText entries.

template class

A C++ class that is instantiated by providing a specific data (or class) type to a template. The compiler builds a class to process data of that type according to the specifications of the template. The Microsoft Foundation Class Library uses template classes to implement the standard collection classes.

temporary object

An object that is created when needed and destroyed after the reference object to which it is bound is destroyed.

temporary window

A window that an application creates for some temporary purpose. For example, a dialog box is a temporary window created to receive user input.

termination

The ending of a thread, process, or program.

termination functions

Functions called internally by MFC member functions when there is a fatal error, such as an uncaught exception that cannot be handled. In MFC, **AfxAbort** is the default termination function. The C/C++ Run-Time Library provides the _abort() function for non-MFC code. Most dynamic-link libraries register termination functions as well.

termination handler

A mechanism by which a developer ensures that a block of termination code is executed, so that resources such as memory, handles, and files are properly closed regardless of how a section of code finishes executing. A termination handler consists of a guarded body of code and a termination block. See also C++ exception handling, structured exception handling (SEH), **try** block.

ternary operator

An operator that takes three operands—for example, the conditional-expression (`? :`) operator in C/C++. See also [binary operator](#), [unary operator](#).

text editor

A program used to manage, edit, and print text files.

text file

A human-readable file composed of text characters. A text file is usually identified by a file extension of .TXT. See also binary file, rich-text format (.RTF) file.

text mode

One of two modes for file I/O operations specified in the file-opening function. In text mode, control characters that specify the end of a line are normalized during I/O operations. In binary mode, no translation occurs.

text-alignment flag

An indicator that determines how text output functions position a string of text on a display or device.

thread

The basic entity to which the operating system allocates CPU time. A thread can execute any part of the application's code, including a part currently being executed by another thread. All threads of a process share the virtual address space, global variables, and operating-system resources of the process.

thread local storage

A Win32 mechanism that allows multiple threads of a process to store data that is unique for each thread. For example, a spreadsheet application can create a new instance of the same thread each time the user opens a new spreadsheet. A dynamic-link library that provides the functions for various spreadsheet operations can use thread local storage to save information about the current state of each spreadsheet (row, column, and so on).

thread switch

A change of context from one thread to another, either inside a single process or across processes.

three-state check box

A square box button control that can have one of three states, usually checked, unchecked (cleared), or indeterminate (grayed).

throw expression

In C++, a statement that transfers program control to a catch block in order to handle an exception.
See also C++ exception handling, catch block, try block.

thumbnail representation

1 Or thumbnail view. In OLE, the reduced image of a document stored within an OLE compound file.

2 In general, a greatly reduced version of an image that contains just enough detail for the image to be recognizable. Thumbnails are often used in a gallery view to allow the user to browse and select from a collection of images.

thunk

A small section of code that performs a translation or conversion during a call or indirection. For example, a thunk is used to change the size or type of function parameters when calling between 16- and 32-bit code.

time-out value

Or time-out delay. The maximum amount of time one entity will wait for another entity to complete a transaction. For example, in ODBC a query time-out value determines the amount of time the database engine will wait for a query's action to complete.

timer identifier (ID)

A value that identifies a timer or the events associated with a timer.

timestamp

A value that specifies the time data was created, modified, accessed, or received. In files, the timestamp may also specify when the data was committed to disk.

TLS

A Win32 mechanism that allows multiple threads of a process to store data that is unique for each thread. For example, a spreadsheet application can create a new instance of the same thread each time the user opens a new spreadsheet. A dynamic-link library that provides the functions for various spreadsheet operations can use thread local storage to save information about the current state of each spreadsheet (row, column, and so on).

token

- 1** In a source program, the basic element recognized by a compiler. Keywords, identifiers, constants, string literals, and operators are examples of tokens.
- 2** A group of security attributes created when a user logs on to the operating system. See also access token, primary token, impersonation token, privilege, security identifier (SID).

tool tip

A tiny pop-up window that presents a short description of a toolbar button's action. Tool tips are displayed when the user positions the mouse over a button for a period of time.

toolbar

A control bar based on a bitmap that contains a row of button images. These buttons can act like pushbuttons, check boxes, or radio buttons. See also dialog bar, status bar

top-level window

A window that has no parent window, or whose parent is the desktop window.

topmost window

The window that overlaps all the other windows even if it is not the active or foreground window.

trace message

Or trace output. An error or diagnostic message employed in debugging to provide information about where in the program execution a problem occurred. In some cases, trace output can provide advance warning about problems that are about to occur.

tracker

In OLE, a border, or adornment, for OLE items that provides a visual cue about the current status of the item. By using different tracker styles, OLE items can be displayed with hatched borders, resize handles, or a variety of other visual effects.

tracking

- 1** In user-interface control, to cause an on-screen displayed symbol, such as a pointer, to match the movements of a mouse or other pointing device.
- 2** In data management, to follow the flow of information through a manual or an automated system (a tracking tool).
- 3** In data storage and retrieval, to follow and read from a recording channel on a disk or a magnetic tape.
- 4** In general, the act of following a path.

tracking size

The window size (maximum or minimum) that the user can produce by dragging a sizing border or splitter bar.

trail byte

In a double-byte or multibyte character set, the second byte of a two-byte character. See also lead byte.

transacted mode

A file-access mode that buffers all changes to a document and writes the changes to disk or discards them only when an explicit commit or revert request occurs. In this way, the original file can be reverted to. See also direct mode, rollback, transaction.

transaction

In data management, a means of completing an all or nothing series of changes to a file. If one change fails, or if there is a system failure during the transaction, the file reverts back to its original state before the transaction began. See also rollback.

transient

In Java, a variable type qualifier denoting that the indicated variable is not part of the persistent state of the object.

translation

- 1** In programming, to convert a program from one language to another—for example, to convert C source code to object code.
- 2** In graphics, to move an image horizontally, vertically, or both, without rotating the image.
- 3** More generally, to convert from one form to another—for example, to translate a scan code into a key code.

translation phase

One of the steps a compiler follows in creating an executable program. In C/C++, these steps include character mapping, line splicing, tokenizing, preprocessing, character-set mapping, string concatenating, translating, and linking.

translation unit

- 1 A single source file, together with all of its include files, supplied as input to the compiler, which combines and translates the files to produce an object file. See also compilation unit.
- 2 In C/C++, a sequence of tokens that the compiler generates during the preprocessor phase. The translation unit incorporates code from the preprocessor directives, such as the **#include** and **#define** directives, into the source code from a single file and excludes any code removed by conditional compilation directives.

Transport Control Protocol/Internet Protocol

A set of transport protocols for the Internet that provides both connection-oriented (TCP) and connectionless (IP) data transfer. Commonly made up of four protocols: IP, TCP, UDP, and ICMP. See also transport protocol, User Datagram Protocol (UDP).

transport layer

- 1** In remote debugging, a data link established between the host machine and the target machine. See also serial transport layer.
- 2** The layer in the ISO/OSI communications model that is responsible for quality of service and accurate delivery of information. Among other services, the transport layer handles error detection and correction.

transport protocol

A set of conventions that govern how data is transported across networks. In a connection-oriented transport protocol, such as Transmission Control Protocol (TCP), applications are required to establish a virtual circuit before data transfer can take place. In a connectionless transport protocol, such as User Datagram Protocol (UDP), an established circuit is not required for data transfer and an application need only open and bind a socket in order to send and receive data.

trigraph

In C/C++, a sequence of two question marks followed by a punctuation character, which the compiler replaces with another character. For example, the compiler will replace the trigraph ??- with the character ~. Trigraphs allow C programs to be written using only the ISO Invariant Code Set, which is a subset of the 7-bit ASCII character set.

TrueType font

A scalable outline font whose glyphs are stored as a collection of line and curve commands plus a collection of hints. Windows uses the line and curve commands to define the outline of the glyph and uses the hints to adjust the length of the lines and the shapes of the curves to correct irregularities in their shapes that occur during rasterization. See also raster font, vector font.

try block

A guarded body of code in a **try-except** frame-based exception handler or **try-finally** termination handler. See also catch block, throw expression.

type cast

An explicit conversion of a variable, structure, object, or expression from one data type to another.

type checking

The examination by a compiler or interpreter of the operations in a program to make sure that the correct data types are being used. See also run-time type information (RTTI).

type declaration

A declaration in a program that specifies the characteristics of a new data type, usually by combining more primitive existing data types. See also data declaration.

type definition

- 1 A declaration that introduces a name which, within its scope, becomes a synonym for a type or a derived type. A type definition is usually used to construct shorter or more meaningful names for types already declared or to encapsulate implementation details that may change.
- 2 A definition that describes the characteristics of an object—for example, a Windows type definition describes the dimensions, colors, behavior, and position for an object of a particular window class.

type library file

Or OLE library. An OLE compound document file containing standard descriptions of data types, modules, and interfaces that can be used to fully expose objects for OLE Automation. The type library file usually has a .TLB filename extension and can be used by other applications to get information about the automation server.

type modifier

A keyword that modifies the data type that follows—for example, **unsigned** can be used to modify an integral data type such as **int**.

type qualifier

A keyword that provides specific properties to an identifier. The **const** type qualifier declares an object to be nonmodifiable. The **volatile** type qualifier declares an item whose value can legitimately be changed by something beyond the control of the program in which it appears, such as a concurrently executing thread.

type safety

The assurance that a given function will be not presented, at run time, with data of a type it cannot handle. Type safety is assured through type checking and/or by the use of template classes that are designed to operate on data of many types. See also run-time type information (RTTI).

type-safe collection

A collection class that enforces type safety on data or objects. For example, a type-safe collection can be implemented by using one of the MFC template-based classes such as **CArray** or **CList**, which can store data of any type.

typed pointer

A pointer to a specified type.

typeface name

The name of a font—for example, Times New Roman.

UDP

A connectionless transport protocol that forms a user interface to the Internet Protocol (IP).

UDT

In OLE, a set of interfaces that allow data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data.

unary operator

An operator that takes only one operand—for example, the increment (**++**) operator. See also binary operator, ternary operator.

unbalanced parentheses

The number of opening parentheses does not equal the number of closing parentheses.

UNC

The standard format for paths that include a local area network file server, as in \server\share\path\filename.

undecorated name

In C++, the form that an identifier has in the source code (as a string of human-readable characters) as opposed to its decorated name, which consists of symbols that are meaningful only to the compiler and the linker.

underflow

Or loss of precision. A condition in which a mathematical calculation produces a result too near to zero to be represented by the range of binary digits available to the computer for holding that value in the specified precision. See also loss of significance.

underhang

The amount of white space between the edge of a character cell and the black part of the glyph.
See also ABC width, overhang.

undo flag

A value, maintained by the edit control, that indicates whether an application can reverse the most recent operation on the edit control (to undo a text deletion, for example).

Unicode

A 16-bit character set capable of encoding all known characters and used as a worldwide character-encoding standard. Windows NT uses Unicode exclusively at the system level.

Uniform Data Transfer

In OLE, a set of interfaces that allow data to be sent and received in a standard fashion, regardless of the actual method chosen to transfer the data.

Uniform Resource Identifier

A variant of Uniform Resource Locator (URL).

Uniform Resource Locator

The address of a resource on the Internet. URL syntax is in the form

`protocol://host/localinfo`, where `protocol` specifies the means of fetching the object (such as HTTP or FTP), `host` specifies the remote location where the object resides, and `localinfo` is a string (often a file name) passed to the protocol handler at the remote location. Also called Universal Resource Locator, Uniform Resource Identifier (URI).

union

- 1** In C/C++, a user-defined data type that can hold values of different types at different times. It is similar to a structure except that all of its members start at the same location in memory. A union variable can contain only one of its members at a time. The size of the union is at least the size of the largest member.
- 2** The keyword used to define union variables.
- 3** Two or more objects joined into one.

Universal Naming Convention

The standard format for paths that include a local area network file server, as in \server\share\path\filename.

universally unique identifier

Or globally unique identifier (GUID). A 128-bit value that uniquely identifies objects such as OLE servers, interfaces, manager entry-point vectors, and client objects. Universally unique identifiers are used in cross-process communication, such as RPC, and OLE.

unresolved external

An error report that results from the linker not being able to identify and link to a data type or function that is used in source code. Unresolved externals usually mean that a DLL or object file was missing during linking.

unsigned integer

A data type that can only hold a whole number with a value greater than, or equal to, zero. In this implementation, the maximum value that an unsigned integer can hold is 0xFFFFFFFF (4,294,967,295).

update handler function

A function that administers changes made to an object or data file to make it more current.

update option

A choice that is made about when and how a system, object, or data file will be changed to make it more current. For example, OLE linked objects, or items, can be automatically updated whenever possible, when the source document is saved, or only on request from the client application.

update region

Or invalid region. In Windows, identifies the portion of a window that is out-of-date or invalid and in need of repainting.

updates

Actions or processes that make a system, object, or data file more current.

upper bound

The upper limit in an allowable range of values.

URI

A variant of Uniform Resource Locator (URL).

URL

The address of a resource on the Internet. URL syntax is in the form

`protocol://host/localinfo`, where `protocol` specifies the means of fetching the object (such as HTTP or FTP), `host` specifies the remote location where the object resides, and `localinfo` is a string (often a file name) passed to the protocol handler at the remote location. Also called Universal Resource Locator, Uniform Resource Identifier (URI).

Usenet

Another name for Internet Newsgroups. A distributed bulletin board system running on news servers, UNIX hosts, online services and bulletin board systems. Collectively, all the users who post and read articles to newsgroups. The Usenet is international in SCOPE and is the largest decentralized information utility, including government agencies, universities, high schools, and organizations of all sizes as well as millions of stand-alone PCs.

User Datagram Protocol

A connectionless transport protocol that forms a user interface to the Internet Protocol (IP).

User Network

Another name for Internet Newsgroups. A distributed bulletin board system running on news servers, UNIX hosts, online services and bulletin board systems. Collectively, all the users who post and read articles to newsgroups. The Usenet is international in SCOPE and is the largest decentralized information utility, including government agencies, universities, high schools, and organizations of all sizes as well as millions of stand-alone PCs.

user-defined message

Any message that is not a standard Windows message.

user-interface object

In Windows, an object that provides functionality to the user-interface. For example, menu items, toolbar buttons, and accelerator keys are all user-interface objects.

user-interface thread

In Windows, a thread that handles user input and responds to user events independently of threads executing other portions of the application. User-interface threads have a message pump and process messages received from the system. See also worker thread.

USRDLL

A version of the Microsoft Foundation Class Library used to statically link MFC into a stand-alone user DLL so that the user DLL can incorporate some of the MFC classes.

UTC

A global time standard equivalent to Greenwich mean time (GMT).

UUID

Or globally unique identifier (GUID). A 128-bit value that uniquely identifies objects such as OLE servers, interfaces, manager entry-point vectors, and client objects. Universally unique identifiers are used in cross-process communication, such as RPC, and OLE.

validation rules

A set of functions that dialog data verification (DDV) employs to validate user entries in a dialog control. For example, the function `DDV_MaxChars` could validate that the string entered in the text-box control is not greater than a specified length.

variable

In programming, a named storage location capable of containing a certain type of data that can be modified during program execution. See also data structure, data type.

variant

- 1** In OLE Automation, an instance of the VARIANT datatype, which can represent values of many different types, such as integers, floats, booleans, strings, pointers, etc.
- 2** In general, one of two or more data elements, functions, libraries, or whatever, exhibiting (usually slight) differences.

VBX

A custom control that can be used in Visual Basic as well as other Microsoft programming systems, including Visual C++. VBX controls are 16-bit dynamic-link libraries and are not supported in 32-bit applications. Visual Basic control files have a .VBX filename extension.

vector font

A scalable font in which the characters are drawn in arrangements of line segments rather than arrangements of curves or bits. Vector fonts are often used in applications that are optimized for output to plotters instead of printers. See also raster font, TrueType font.

Veronica

A database of the names of almost every menu item on thousands of gopher servers. This database can be searched from most major gopher menus. See also Archie.

version information

An application's company and product identification, product release number, and copyright and trademark notification. Version information is held in a standard form in the executable (.EXE) file or dynamic-link library (DLL) file and is accessible by various tools and Windows functions.

version resource

A resource that contains either text or binary data about an application's version information.

Very Easy Rodent-Oriented Net-wide Index to Computerized Archives

A database of the names of almost every menu item on thousands of gopher servers. This database can be searched from most major gopher menus. See also Archie.

view

- 1 (noun) A window object through which a user interacts with a document.
- 2 Or aspect. (noun) Generally, the manner in which data or a graphical image is displayed.
- 3 (verb) To display information on a computer screen, as in to view a file.

viewer

Program for opening image files and files in other special formats, such as audio and video.

viewport

A rectangle in device space that is used to specify a transformation between page and device space. The viewport extents (height and width) are always measured in pixels for a video display or in dots for printers.

viewport origin

The corner of the viewport from which the height and width of the viewport are measured.

VIRTKEY

In an accelerator table resource, a flag used to indicate that the keystroke value in the definition is a virtual key code.

virtual base class

A base class whose derived classes contain only one shared instance of its members even if the indirectly derived classes have done multiple inheritance. The class declaration for the derived class must have the keyword **virtual** before the base class specifier in order for it to be a virtual base class. In the following example, class `MultiC` has only one subobject of class `Base` even though it has inherited from more than one class derived from `Base`:

```
class Base {    };  
class DerivedA: virtual public Base {    };  
class DerivedB: virtual public Base {    };  
class MultiC: public DerivedA, public DerivedB {    };
```


virtual destructor

A destructor implemented by declaring a base class's destructor with the keyword `virtual`. A virtual destructor ensures that, when `delete` is applied to a base class pointer or reference, it calls the destructor implemented in the derived class, if an implementation exists.

virtual device driver

A low-level software component that manages a single resource, such as a display screen or a serial port, on behalf of all running processes. A VxD is always 32-bit protected mode code and is frequently written in assembly language.

virtual function

A member function of a base class, where the function is declared with the keyword `virtual`. If a base class contains a virtual function and a derived class defines the same function, the function from the derived class is invoked for objects of the derived class, even if it is called using a pointer or reference to the base class.

virtual inheritance

In C++, a method of deriving classes by preceding the name of the base class with the keyword **virtual**. With virtual inheritance, indirectly derived classes contain only one shared instance of the base class's members even if they have done multiple inheritance. See also virtual base class.

virtual key code

In Windows, a device-independent value translated from the scan code by the keyboard device driver. The code identifies the keyboard key. See also VIRTKEY.

virtual memory

A memory-management scheme that allows an application to see a large, continuous block of primary memory (RAM) that, in reality, is a much smaller block of primary memory supplemented by secondary memory (such as a hard disk), with blocks of data being moved into and out of primary memory. See also swap file.

Virtual Reality Modeling Language

A vector-based language for modeling three-dimensional environments. It sends ASCII text files over the Internet, which are translated by the VRML viewing engine at the other end. VRML complements HTML.

visible region

In Windows, the portion of a window visible to the user.

Visual Basic control

A custom control that can be used in Visual Basic as well as other Microsoft programming systems, including Visual C++. VBX controls are 16-bit dynamic-link libraries and are not supported in 32-bit applications. Visual Basic control files have a .VBX filename extension.

void pointer

A pointer to an object of unknown type. See also typed pointer.

volatile

1 In C/C++ and J++, a type qualifier that declares an item whose value can legitimately be changed by something beyond the control of the program in which it appears, such as a concurrently executing thread.

2 More generally, describes information that is stored in memory and not preserved when a process or application terminates.

VRML

A vector-based language for modeling three-dimensional environments. It sends ASCII text files over the Internet, which are translated by the VRML viewing engine at the other end. VRML complements HTML.

VxD

A low-level software component that manages a single resource, such as a display screen or a serial port, on behalf of all running processes. A VxD is always 32-bit protected mode code and is frequently written in assembly language.

watch expression

An expression that can be evaluated as the program progresses and can be viewed in the Watch window during debugging.

wave file

A Microsoft standard file format for storing waveform audio data. Wave files have a .WAV filename extension.

waveform audio

A technique of re-creating an audio waveform from digital samples of the waveform.

weak external

A special type of external reference that allows the linker to fix up an address with an alternate symbol. By using a weak external record, a library developer can provide a method for the linker to use an alternate symbol if the code does not make any other references to the module in which the primary symbol resides.

Web browser

Software that renders Web pages on a local computer. See also World Wide Web.

what you see is what you get

A display method (pronounced WI-zy-wig) that shows documents and graphics as they will appear when printed.

white space

- 1** Spaces, tabs, and blank lines. White space sometimes refers to the C/C++ language-specific character constants that create white spaces; namely, ' ', '\t', and '\n'.
- 2** The empty areas in a window or the margins on a document.

wide character

Or Unicode character. A 2-byte multilingual character code. Any character, including technical symbols and special publishing characters, can be represented as a wide character. Note that a multibyte character or a double-byte character can be 2 bytes in size, but neither is a wide, or Unicode, character. See also wide-character constant.

wide-character constant

A 16-bit character constant that can specify a member of the extended execution character set. Wide-character constants can be used to express characters in alphabets that are too large to be represented by type **char**. Use wide-character constants in place of multicharacter constants to ensure portability. Syntactically, a wide-character constant is a character constant prefixed by the letter **L**.

WIN.INI

A text file that stores initialization information for Windows and Windows-based applications. The WIN.INI file is configured with a number of sections and entries, depending on a particular system's hardware and software requirements. For example, entries in the [Desktop] section control the appearance of the desktop and the position of windows and icons.

Win32 platform

A platform that supports the Win32 API. These platforms include Intel Win32s, Windows NT, Windows 95, MIPS Windows NT, DEC Alpha Windows NT, and Power PC Windows NT.

window characteristic

An attribute of a window, such as size, position, or the presence of scroll bars.

window class

A set of attributes that Microsoft Windows uses as a template to create a window in an application. Windows requires that an application supply a class name, the window-procedure address, and an instance handle. Other elements may be used to define default attributes for windows of the class, such as the shape of the cursor and the content of the menu for the window.

window extent

The width (x-extent) or height (y-extent) of a window.

window handle

In the Win32 API, a 32-bit value (assigned by Windows) that uniquely identifies a window. An application uses this handle to direct the actions of functions to the window. A window handle has the **HWND** data type; an application must use this type when declaring a variable that holds a window handle.

window origin

- 1** The upper-left corner of a window's client area.
- 2** The corner of the window from which the extents are measured. See also viewport origin.

window procedure

A function, called by the operating system, that controls the appearance and behavior of its associated windows. The procedure receives and processes all messages to these windows.

window rectangle

The coordinate rectangle that encloses the entire window. The window rectangle is defined by two ordered pairs of numbers that define the upper-left and lower-right corners of the rectangle. See also client area.

window style

A named constant that defines an aspect of the window's appearance and behavior not specified by the window's class.

window-management function

A function that carries out an action on a window. For example **SetWindowPos** is a window-management function that can change the size, position, or Z order of a window.

Windows initialization file

A text file that stores initialization information for Windows and Windows-based applications. The WIN.INI file is configured with a number of sections and entries, depending on a particular system's hardware and software requirements. For example, entries in the [Desktop] section control the appearance of the desktop and the position of windows and icons.

Windows Open Services Architecture

A specification for an open set of API functions that allow Windows-based desktop applications to connect to multiple computing environments, with a single version of each application able to work with multiple platforms. With WOSA, applications can access available information regardless of the type of network in use, the types of computers involved, or the types of back-end services available.

Windows Write file

A document file that is associated with the Windows Write text editor. The default filename extension for Windows Write files is .WRI.

wizard

A special form of user assistance that guides the user through a difficult or complex task within an application. For example, a database program can use wizards to generate reports and forms. In Visual C++, the AppWizard generates a skeleton program for a new C++ application.

word alignment

Data in memory that is aligned on word boundaries. In 32-bit systems, word alignment means that the first byte of a data object is located at an address that is a multiple of 4.

word boundary

A memory address that is a multiple of the machine's word size, in bytes. See also word alignment.

wordwrap

To break lines automatically in order to keep the text within the boundaries of a window or the margins of a document.

worker thread

A thread that handles background tasks while the user continues to use an application. Tasks such as recalculation and background printing are examples of worker threads. See also user-interface thread.

working set

- 1 The set of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault. If free memory in the computer is above a threshold, pages are left in the working set of a process even if they are not in use. When free memory falls below a threshold, unneeded pages are trimmed from working sets.
- 2 The average amount of memory, either physical or virtual, used by the process. The longer a process is running, the more accurate this value is.

workspace

A window or task-management technique that consists of a container holding a set of objects, where the windows of the contained objects are constrained to a parent window. A workspace is similar to the multiple document interface, except that the windows displayed within the parent window are of objects that are actually contained in the workspace.

workspace configuration file

A file created and maintained by the Developer Studio. The file contains all the information about the project workspace, including a list of all the projects, their associated source files, each project's output file, the settings and tools required to build each project, as well as the physical layout and characteristics of Developer Studio: window layout and characteristics, syntax coloring, editor preferences, and so on.

workspace window

In the Microsoft Developers Studio, a window that displays graphical representations of a project workspace. The panes in the window display projects and files, classes for C++ files, and the table of contents for the online help system.

World Wide Web

Or the Web. The portion of the global Internet that uses hypertext links to connect pages and resources in a way that lets you reach any page from any other page. Web browsers enable you to navigate the Web.

WOSA

A specification for an open set of API functions that allow Windows-based desktop applications to connect to multiple computing environments, with a single version of each application able to work with multiple platforms. With WOSA, applications can access available information regardless of the type of network in use, the types of computers involved, or the types of back-end services available.

wrapper class

A C++ class whose function is to provide an alternative interface for objects of another class. For example, C exceptions can be handled as typed exceptions, with the C++ catch handler, by using a C exception wrapper class.

wrapper function

A function whose purpose is to provide an interface to another function. Such an interface might create type safety where none existed in the original function.

write-only

An attribute of an object (a C++ **ostream** object, for example), a pipe, or a file specifying that data can only be written to it, not read from it.

WWW

Or the Web. The portion of the global Internet that uses hypertext links to connect pages and resources in a way that lets you reach any page from any other page. Web browsers enable you to navigate the Web.

WYSIWYG

A display method (pronounced WI-zy-wig) that shows documents and graphics as they will appear when printed.

x-extent

The width of a window. See also y-extent.

y-extent

The height of a window. See also x-extent.

Z order

Indicates a window's position in a stack of overlapping windows. This window stack is oriented along an imaginary axis, the z-axis, extending outward from the screen. The window at the top of the Z order overlaps all other windows. The window at the bottom of the Z order is overlapped by all other windows. An application sets a window's position in the Z order by placing it behind a given window or at the top or bottom of the stack.

zooming

Enlarging a selected window or portion of a graphical image in order to see finer details in the image.

Java Hierarchy Chart

{ewc msdncd, EWGraphic, hie0a 0 /a "hier6JAVAH.SHG"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Introduction

The Microsoft Visual J++™ version 1.0 Development System for Microsoft Windows® 95 and Microsoft Windows NT™ (versions 4.0 and later) is an integrated development environment for creating Java applets and applications.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Microsoft Developer Studio

Microsoft Developer Studio is the development environment for Visual J++. It consists of a set of tools that run under Microsoft Windows 95 or Microsoft Windows NT (versions 4.0 and later). Developer Studio gives you the tools to complete, test, and refine your program. It includes a text editor, resource editors, project build facilities, an integrated debugger, and Books Online. You can control the operation of all the tools from a single, integrated environment. Because these tools run under Windows, they use a variety of familiar methods in their operation. For example, you can select a variable name in an editor window while debugging and drag that name into the Watch window. The debugger then evaluates the variable and displays the result in the Watch window. Developer Studio also includes toolbars so you can quickly invoke commands by clicking a button. To help you choose the correct button, each one displays a descriptive label if the mouse pointer rests on it. If the default toolbars are not to your liking, you can customize them or create your own toolbars with the toolbar buttons of your choice.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Time-Saving Wizards

Visual J++ provides some powerful tools:

Applet Wizard

Applet Wizard generates source files and .GIF files based on classes in the Java libraries. By selecting options in Applet Wizard, you can customize the starter files that Applet Wizard generates. Once you have selected your applet's options in Applet Wizard, Visual J++ builds an applet from those starter files, without any further work on your part.

Resource Wizard

The Resource Wizard can be invoked after a resource template has been created. The Resource Wizard may also be used on .RES files from existing Windows project. The Resource Wizard evaluates the resources in the resource template or .RES files and generates Java code based on dialog and menu resources it encounters. The code generated is placed in class files and layout files that can be added to a Visual J++ project. Projects containing the Resource Wizard-generated Java code, when executed, will create dialogs and menus comparable to those described by the template.

To build an applet with Visual J++, run Applet Wizard to create the skeleton of your application, then add new classes, variables, and methods required for your applet by using the ClassView's right mouse pop-up menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Developer Studio Projects

Microsoft Developer Studio organizes development in project workspaces. Each project consists of a set of source files required for an application and one or more configurations for that project. A configuration specifies such things as the platform for which the application is intended, and the tools and settings to use when building. Within a project workspace, one project can be a subproject of another project. This organization creates a dependency relationship used by the build system to automatically keep both the project and the subproject up to date when building the output files. The inclusion of multiple projects with subprojects in a workspace allows you to group, build, and maintain dependencies among related applications. By using multiple configurations, you can extend the scope of a single project but still maintain a consistent source-code base from which to work.

When the Project Settings dialog box is selected from the Build menu, you can quickly set options for any configuration in a project, any file in a configuration, or all the files in a project. If you have file types in your project that the build system does not process by default, you can specify custom commands to process those files.

Developer Studio includes a Project Workspace window which displays various aspects of the projects. In FileView, you can examine relationships among the files contained in the project, and take the appropriate actions on the files. With Visual J++ installed, you can examine classes and their members in ClassView, and quickly display a class hierarchy, add a member, or open the file containing the class.

Once you have specified the projects in your workspace, the configurations that your project is to build, and the tool settings for those configurations, you can build the project with the commands on the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Build Error Correction

If your build has errors, Developer Studio can help you fix them quickly. The Output window displays a list of errors generated during a build. If you press the F4 key, Developer Studio displays an editor window with the source file and marks the line of code associated with the first error in the Output window so you can immediately correct the code. With menu commands and keyboard shortcuts, you can then move quickly to the next or previous error.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Integrated Debugger

After you have corrected all the build errors, you can use the integrated debugger to correct logic errors. The debugger allows you to monitor your program as it runs and to stop it at locations or situations of your choosing. You can set a breakpoint on a particular line of code, for instance, and have your application execute until it reaches that line.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


User Preferences

With Microsoft Developer Studio you can customize its operation to suit your preferences. You can select fonts, specify colors for particular types of text, and change the size of text in a window. For example, in text editor windows, you can display language elements, such as comments or keywords, in the color of your choice. When you establish a layout for the windows associated with a particular project workspace, Developer Studio retains that layout of open files and window positions the next time you start the project. When you are debugging an application, you can choose which windows and toolbars to display, and Developer Studio retains your selections for all subsequent debugging sessions.

If you have some preferences for shortcut keys other than the defaults, you can change any of the shortcut keys to your liking, add shortcut keys, set multiple shortcut keys for a command, and specify the windows in which any shortcut is active. You can use the Keyboard command on the Help menu to display a list of the current keyboard shortcuts and print all or part of the list. Developer Studio also provides keyboard emulations for BRIEF® and Epsilon™ text editors. In a text editor window, you can record keystrokes and then play them back to recreate that sequence of commands.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Extensive Information

Developer Studio provides several methods to learn about the development environment or the supporting software. The *Test Drive* contains a series of scenarios to help familiarize you with Developer Studio, with the methods and processes you need to use within Developer Studio, and with the development of Java programs using Visual J++. From your source code in an editor window, you can readily get information about a Java language element. If you select an element in an editor window, and press F1, Developer Studio displays reference information for that element.

InfoView in the Project Workspace window displays the table of contents for Books Online. Books Online contains the entire Visual J++ documentation set as well as reference information from a number of software development kits. You can browse through the table of contents and select topics to view. From any topic, you can search through all the text of Books Online for the occurrence of a selected word or combination of words with the Search command on the Help menu. If you need information about an open dialog box, you can choose the Help button to view descriptions of its controls, and methods to access further information, if necessary.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Visual J++ User's Guide

In the *Visual J++ User's Guide*, you will find procedures that show you how to undertake various development tasks with Microsoft Developer Studio. The *Visual J++ User's Guide* also includes reference information on underlying command-line tools.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InfoViewer Integrated into Developer Studio

InfoViewer—the new online documentation system—is integrated into Microsoft Developer Studio. The Table of Contents for Books Online appears in the InfoView pane of the Workspace window.

Help topics appear in the InfoViewer topic window (an MDI child window). This integration offers important advantages:

- You can see both your source window and a topic window without having to switch between applications. You have the same layout control over a topic window as you have over a source window or debugger window—you can make it a docking window or position it within the workspace to suit your needs.
- It's easy to copy text from a topic window into a source window, using either the Clipboard or drag-and-drop.

You can customize the way you access Books Online, reassigning keyboard shortcut keys or creating custom toolbars with your own selection of InfoViewer buttons.

For more details, see Finding Information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dockable Windows

You can view Books Online with the topic window at full screen, as an MDI child window, in a floating state, or in a docked position. Try each state to see what suits your work style. If the topic window is an MDI child window, pressing ESC hides it. If the topic window is either floating or docked, pressing ESC+ESC hides it (the first ESC returns the focus to the current source window and the second hides the topic window). Regardless of a topic window's previous state, pressing ALT+1 makes it visible.

To dock and undock your topic window, toggle the Docking View menu item from the topic window's right mouse menu. A check mark to the left of the Docking View menu item indicates that the window is in docking mode.

Try docking your topic window when you are coding. When you press F1 on a keyword, the topic window appears docked, resizing your source window to accommodate the new docked window. If you make your docked window large, you have enough room to view both your code and help. This arrangement also makes drag-and-drop copy from the topic window to your source easy. Pressing ESC+ESC hides the topic window and your source window expands to its original size.

Tips for Using Books Online in a Docked Window

To gain even more room for help text, you can:

- Hide the topic window toolbar by pressing CTRL+SHIFT+T
 - Hide the topic title by pressing CTRL+T
- Pressing CTRL+S will synchronize the current topic with its title in the InfoView pane if you want to keep the topic title hidden.

Double-click on the toolbar of a docked window to change it to a floating state. In a floating state, you can move the window to dock on any side of the application frame. You can also move a floating window completely outside of Developer Studio to the desktop itself. Double-click on the caption bar or toolbar of a docked window to return the window to its floating state.

All of the buttons in the toolbar have default access keys. Use your keyboard to move to the next topic, synchronize to the InfoView pane, or display the See Also list.

You can resize the topic title to keep the non-scrolling region small. You can also change the size of the text in the InfoViewer topic window.

{ewl msdncd, EWGraphic, jug1u 0 /a "build.bmp"} To change the size of InfoViewer's topic title or text

- 1 With the cursor in the topic window, click the right mouse button.
The right mouse menu appears.
- 2 Select Options from the right mouse menu.
The Options dialog appears.
- 3 Select the InfoViewer tab.
- 4 Inside the Zoom group box, set the magnification percentage for the topic title and the topic text.

For more details, see Finding Information.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MSDN Compatibility

InfoViewer can display both Books Online and the Microsoft Development Library. If you have installed the Microsoft Development Library, you can view it from Visual J++. You no longer have to use a separate application to get information outside of Books Online.

You can also use the Microsoft Development Library to get F1 Help from a source file. When the Microsoft Development Library is the current Information Title, InfoViewer opens the Microsoft Development Library topics when you press F1 on a keyword in a source file.

For more details, see [Finding Information](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Two Presentation Styles

The InfoViewer lets you view online information in two ways:

- By book
- By topic

You can view information organized as books and topics by looking under the Visual J++ Books node in the Table of Contents.

For more details, see [Finding Information](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Printing Capabilities

Printing capabilities are available in Visual J++ version 1.0. You can now print entire topics, or even entire books, in a single step. Just select what you want to print, and choose the Print command from the File menu.

For more details, see [Finding Information](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Easy Access to Samples

Samples have been integrated into the Books Online, making them easier to find and use. In addition, hard-disk space is saved because the samples now remain on the Visual J++ CD unless you copy them to your hard drive.

Samples are located in a top-level category in the InfoView pane. The behavior of each sample application is described by an abstract. This description makes it easy to browse among the samples or search for the sample or samples that implement a particular piece of functionality. If you want to view the source files or run the program, you can do so without extracting the sample from Books Online. Source files are viewed directly in Developer Studio. If you decide you want to modify and recompile the sample application yourself, you can extract the source files for the desired sample application.

For more details, see [Finding Information](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Online Glossary

Visual J++ Books Online now includes a comprehensive glossary of technical terms. You can get glossary definitions in two ways:

- As topics under the Glossary category (under the Miscellaneous category) in the InfoView pane.
- As pop-ups available by clicking gray terms in Help text.

Terms defined in the glossary are "hot" when they appear in the core Visual J++ documentation set (the Test Drive, User's Guide, Language Specification, and Build Errors categories)—you simply click a term to display the definition in a pop-up window.

Note You can change the color of glossary pop-ups (as well as other jumps) from the Format tab on the Options dialog box in the topic window.

For more details, see Finding Information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Finding Information

Welcome to InfoViewer, the documentation system that's an integrated part of the Microsoft Developer Studio. Click any of the hot-links below for information on using the features of InfoViewer.

**Entering
Books
Online**

[F1
Help](#)

[Index](#)

[Full-
Text
Search
h](#)

[Table
of
Conte
nts](#)

**Custo
mizing
Books
Online**

[Custo
mizing
the
Displa
y](#)

[Using
Subse
ts](#)

[Addin
g
Annot
ations](#)

[Viewin
g
Differe](#)

**Navigating
Books
Online**

[Jumping
to
Related
Topics](#)

[Browsing
Query
Results](#)

[The Back
Button
and the
History
List](#)

[Bookmar
ks](#)

**Special
Features**

[Sample
Program
s](#)

[Copying
and
Printing](#)

[Toolbars](#)

[Keyboard
Shortcuts](#)

[nt](#)
[Inform](#)
[ation](#)
[Titles](#)

[Customiz](#)
[ing](#)
[Keyboard](#)
[/Toolbars](#)
[Tips for](#)
[Using](#)
[InfoView](#)
[er](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Entering Books Online

[Back to Introductory Screen](#)

[F1 Help](#)

[Index](#)

[Full-Text Search](#)

[Table of Contents](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

F1 Help

The fastest way to get help from InfoViewer is with the F1 key. Simply position the insertion point on the word you want information on and press F1. This works in source windows, so you can get information on a keyword or an API method, and the output window, so you can get information on error messages. It's also available from InfoViewer Topic windows if you select a word with the mouse.

{ewl msdncd, EWGraphic, jug2v 0 /a "build.bmp"} To get help on a word in a text window

- 1 In the text window, position the insertion point on the word that you want help on. Press F1.
If there are multiple topics in the information title that are indexed under the word you requested, the Select Reference dialog box appears, displaying all the topics found. If there is only one topic indexed under that word, that topic appears immediately.
- 2 Double-click the topic you want to look at. You can also select the topic and press the Display button or press the ENTER key.
A Topic window appears, displaying information about the keyword you wanted help on.

If there are too many topics indexed under the keyword, you can instruct InfoViewer to use only a portion of its contents, or a "subset," when responding to F1 requests. See Using Subsets.

If you have multiple information titles (.MVB files) installed, occasionally the word you asked for help on may have entries in two or more information titles. In such a situation, InfoViewer lists the names of the information titles and lets you choose which one you want to view.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Index

Besides the F1 key, the fastest way to get information is to look in the index. InfoViewer provides an index to all the topics, similar to the index in a printed book. Every topic is indexed under one or more keywords.

{ewl msdncd, EWGraphic, jug3v 0 /a "build.bmp"} To search the index

- 1 Click the right mouse button inside an InfoViewer Topic window and choose Search from the pop-up menu. (If no Topic window is open, pull down the Help menu and choose Search from there.) The default keyboard shortcut is CTRL+F when a Topic window has focus.
- 2 Select the Index tab in the Search dialog box.
- 3 Type the word you want information on. Notice that the list displays the portion of the index that matches the word you've typed. You can also select a word from the index directly. Press the List Books button or press the ENTER key. You can also double-click an entry in the index.

The list box at the bottom of the dialog box displays all the topics indexed under the keyword you specified.

- 4 Select the topic you want to look at, and then press the Display button. You can also double-click the entry in the list box.

A Topic window appears, displaying information about the keyword you entered.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Full-Text Search

If you can't find the information you're looking for in the index, InfoViewer also offers full-text search capability in the form of queries. This finds every occurrence of a given word or phrase, anywhere within the information title, or within a subset of the contents that you specify.

{ewl msdncd, EWGraphic, jug4v 0 /a "build.bmp"} To perform a query (full-text search)

- 1 Click the right mouse button inside an InfoViewer Topic window and choose Search from the pop-up menu. (If no Topic window is open, pull down the Help menu and choose Search from there.) The default keyboard shortcut is CTRL+F when a Topic window has focus.
- 2 Select the Query tab in the Search dialog box. For more about queries, see Basic Query Syntax.
- 3 Type the word or phrase you want to find. If you're searching for a phrase, delimit the phrase with quotation marks. Press the Query button or press the ENTER key.
The Query Results window appears, listing all the topics containing the word or phrase you specified. If your query produced too many hits, see Narrowing the Focus of Your Query.
- 4 Double-click the entry you want to look at, or select it using the cursor keys and press the RETURN key.
An InfoViewer Topic window appears, displaying the topic you selected. Search hits are highlighted in the topic text. (To toggle highlighting on and off, click the right mouse button inside a Topic window and choose Highlights from the pop-up menu; the default keyboard shortcut is CTRL+H. To ensure that the first hit is displayed when a topic is opened, click the right mouse button inside a Topic window and choose Options. The Options dialog box appears, opened to the InfoViewer tab; select the "Jump to First Highlight" check box.)

You can keep the Query Results window open by "pinning it"; click the push-pin button near the upper left corner.

You can also search for a word in the current topic by typing the word in the Find box on the Standard toolbar. If you press ENTER, InfoViewer will highlight occurrences of that word in the current topic. If you press SHIFT+ENTER, InfoViewer performs a query (full-text search).

To navigate within the query results, see Browsing Query Results.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Narrowing the Focus of Your Query

You can choose the scope of your query (full-text search). The default scope is the entire contents of the information title, but you can also choose to search one of the following areas:

- The current topic. This lets you search for occurrences of a phrase in a topic. This is most useful when you have search highlighting turned on. (To toggle highlighting on and off, click the right mouse button inside a Topic window and choose Highlights from the pop-up menu; the default keyboard shortcut is CTRL+H.)
- The topics found as a result of the last search. This lets you narrow the focus of your search in successive steps.
- A subset of the contents. This lets you search just selected books, instead of the entire contents of help. You can do this from the Select Default Subsets dialog box; from the Help menu, choose Set Default Subsets. For more information, see Using Subsets.

You can also choose the topic area to search. The default is the full text of the topic (including the title), but you can also choose to search just the topic titles.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Basic Query Syntax

A query consists of the word or phrase you want to find. See Words, Phrases, and Wildcards. You can use Boolean operators (AND, OR, NOT) and the proximity operator (NEAR) to specify additional search information. See Operators: AND, OR, NOT, and NEAR.

The basic rules for formulating queries are as follows:

- Queries are case-insensitive, so you can type your query in uppercase or lowercase.
- You may search for any combination of letters (a-z) and numbers (0-9) except for the words in the exception list (a, an, and, as, at, be, but, by, do, for, from, have, he, in, it, not, of, on, or, she, that, the, there, they, this, to, we, which, with, you) which are ignored during a search.
- Punctuation marks such as the period (.), colon (:), semicolon (;), comma (,), and hyphen (-) are ignored during a search.
- Use single or double quotation marks to delimit phrases; you cannot search for quotation marks.

For more information, see Nested Expressions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Words, Phrases, and Wildcards

You can search for a word or a phrase in a query, and you can include wildcards in the search.

To search for	Example	Results
A single word	select	Topics that contain the word "select" (you will also find its grammatical variations, such as "selector" and "selection.")
A phrase	"new operator" or 'new operator'	Topics that contain the literal phrase "new operator" and all its grammatical variations. Without the quotation marks, the query is equivalent to specifying new AND operator or which will find topics containing the individual words, not the phrase.
A wildcard expression	esc*	Topics that contain the terms "ESC," "escape," "escalation," and so on.

The
asterisk
cannot be
the only
character in
the term.

80? Topics that
86 contain the
terms
"80186,"
"80286,"
"80386,"
and so on.
The
question
mark
cannot be
the only
character in
the term.

*86 Topics that
contain the
terms
"386,"
"486,"
"x86,"
"QEMM386
, "8086,"
and so on.
The
asterisk
cannot be
the only
character in
the term.

Note The search engine uses stemming (finding minor grammatical variations of a word) unless your search term contains a number. If a search term contains a number, stemming is disabled. Quotation marks do not disable stemming.

For example:

```
ad fin "add," "adds,"
d ds "added"
wi fin "win32s"
n3 ds
2s
wi fin "win32"
n3 ds
2
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Operators: AND, OR, NOT, and NEAR

You can use Boolean and proximity operators to combine expressions and create more precise queries.

To search for	Example	Results
Both terms in the same topic	dib AND palette -or- dib & palette	Topics containing both the words "dib" and "palette."
Either term in a topic	raster OR vector -or- raster vector	Topics containing either the word "raster" or the word "vector."
The first term without the second term	ole NOT dde -or- ole ! dde	Topics containing the word "OLE," but not the word "DDE."
Both terms in the same topic, close together	user NEAR kernel	Topics containing the word "user" within eight words of the word "kernel."

{ewl msdncd, EWGraphic, jug8v 0 /a "build.bmp"} To change the NEAR operator's proximity setting

- 1 Click the right mouse button inside an InfoViewer Topic window and choose Options from the pop-up menu.
The Options dialog box appears, opened to the InfoViewer tab.
- 2 In the NEAR Means Within edit control, specify how close two words must be to satisfy a query

containing the NEAR operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Nested Expressions

You can use parentheses to nest expressions within a query. The expressions in parentheses are evaluated before the rest of the query.

If a query doesn't contain any nested expressions, it's evaluated from left to right. For example:

pen NOT ole OR dde

finds topics containing the term "pen" without the term "OLE," or topics containing the term "DDE." On the other hand,

pen NOT (ole OR dde)

finds topics containing the term "pen" without either of the terms "OLE" or "DDE."

Nesting allows you to create more complex search expressions. For example:

pen AND ((ole OR dde) NEAR window)

finds topics containing the term "pen" along with the terms "OLE" and "window" close together, or containing "pen" along with the terms "DDE" and "window" close together. You cannot nest expressions more than five levels deep.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Table of Contents

If you'd like to see an overview of what's available in the information title, use the Table of Contents. This shows you the titles of the books and sections that make up the information title, allowing you to browse among the various types and categories of information available.

{ewl msdncd, EWGraphic, jug10v 0 /a "build.bmp"} To view the help Table of Contents

- From the Help menu, choose Contents, or press the InfoView tab at the bottom of the Project Workspace window.

The Workspace window switches to the InfoView pane, which displays the Table of Contents.

{ewl msdncd, EWGraphic, jug10v 1 /a "build.bmp"} To open a book

- You can open a book by clicking the '+' symbol in front of the name or double-clicking the name itself. The keyboard shortcuts are ENTER, RIGHT ARROW or PLUS SIGN when the book is selected.

Notice that the '+' symbol changes to a '-' symbol when a book has been opened. You can also use the UP ARROW or DOWN ARROW to move up or down.

{ewl msdncd, EWGraphic, jug10v 2 /a "build.bmp"} To close a book

- You can close a book by clicking the '-' symbol in front of the name or double-clicking the name itself. The keyboard shortcuts are ENTER, LEFT ARROW or MINUS SIGN when the book is selected.

Notice that the '-' symbol changes to a '+' symbol when a book has been closed. You can also use the UP ARROW or DOWN ARROW to move up or down.

{ewl msdncd, EWGraphic, jug10v 3 /a "build.bmp"} To view a topic

- Double-click its page icon.

The topic you selected is displayed in an InfoViewer Topic window.

If you're interested in only a portion of the contents, you can have the Table of Contents display a subset. See Using Subsets for more information.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Navigating Books Online

[Back to Introductory Screen](#)

[Jumping to Related Topics](#)

[Browsing Query Results](#)

[The Back Button and the History List](#)

[Bookmarks](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Jumping to Related Topics

From any InfoViewer Topic window, there are several ways to get related information.

{ew This [toolbar](#)
c button expands
ms and scrolls the
dnc Table of Contents
d, to show you where
EW the current topic is,
Gr relative to the rest
ap of the Table of
hic Contents. The
jug default [keyboard](#)
12v [shortcut](#) is
0 / CTRL+S. You can
a also have the
"ug synchronization
uid done automatically;
e.B see [Customizing](#)
MP [the Display](#).
"}
}

{ew These [toolbar](#)
c buttons move you
ms to the topic that
dnc precedes or follows
d, the current one, as
EW listed in the Table
Gr of Contents. The
ap default [keyboard](#)
hic [shortcuts](#) are
jug CTRL+SHIFT+P and
12v CTRL+SHIFT+N,
1 / respectively.
a
"ug
uid
e.B
MP
"}
}

{ew This [toolbar](#)
c button returns you
ms to the last topic you
dnc viewed; you can
d, backtrack up to 50
steps. The default

EW [keyboard](#)
Gr [shortcut](#) is
ap CTRL+B. You can
hic, also see a list of
jug previously viewed
12v topics; see [The](#)
2 / [Back Button](#)
a [and the History](#)
"ug [List](#).

uid
e.B
MP
"}
{ew

This [toolbar](#)
c button displays a
ms drop-down list
dnc containing related
d, topics. The default
EW [keyboard](#)
Gr [shortcut](#) is
ap CTRL+SHIFT+S. For
hic, example, click the
jug See Also button for
12v this topic to see a
3 / list of its related
a topics .

"ug
uid
e.B
MP
"}
Wor

These are hot-
ds links. Clicking a
or hot-link can either
phr move you to
ase another topic or
s display a pop-up
that window. The
are default [keyboard](#)
colo [shortcut](#) is the TAB
red key to move to the
next [jump](#) in the
topic text, and the
ENTER key to
activate the [jump](#).
By default, jumps
to other topics are
colored green,

while pop-ups are
colored gray. To
change the
coloring, see

[Customizing the
Display](#).

If you want to find out more about a word that's not a hot-link, you can select it with the mouse and then press the F1 key. This does the same thing as pressing F1 after you've selected a word in a source window: it searches the help index for that word and displays the topic indexed under that word.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Browsing Query Results

InfoViewer has several features to help you navigate through the results of your queries.

{ewl msdncd, EWGraphic, jug13v 0 /a "build.bmp"} To see the results of your last query

- From the View menu, choose InfoViewer Query Results.
The Query Results window appears.

Double-click the name of a topic to go there directly. You can keep the Query Results list open while you read the topic by “pinning” it; click the push-pin button near the upper left corner.

Search hits are highlighted in topic text. (To toggle highlighting on and off, click the right mouse button inside a Topic window and choose Highlights; the default keyboard shortcut is CTRL+H. To ensure that the first hit is displayed when a topic is opened, click the right mouse button inside a Topic window and choose Options from the pop-up menu. The Options dialog box appears, opened to the InfoViewer tab; select the Jump to First Highlight check box.)

{ewl msdncd, EWGraphic, jug13v 1 /a "build.bmp"} To navigate through the results list

{e If the Query Results
w window is pinned
c open, you can use
m these buttons in its
sd [toolbar](#) to move to
nc the previous or next
d, topic in the results
list.

E
W
G
ra
p
hi
c,
ju
g
1
3v
2
/a
"u
g
ui
d
e.
B
M


```

P"
}
{e If you don't want to
w keep the Query
c results window
m open, display the
sd InfoViewer
nc toolbar. (From the
nc View menu, choose
d, Toolbars. In the
E Toolbars dialog
W box, select the
G InfoViewer check
ra box.) You can use
p these toolbar
hi buttons to move to
c, the next or previous
ju topic in the results
g list.
1
3v
3
/a
"u
g
ui
d
e.
B
M
P"
}

```

By default, the results list is sorted based on the number of hits found in each topic, in descending order; the number in the first column indicates the topic's rank.

{ewl msdncd, EWGraphic, jug13v 4 /a "build.bmp"} To sort the results list

- To sort the results list by book title, click the "Book" button at the top of the second column. To sort the results list by the topic title, click the "Topic" button at the top of the third column.

Note that the Query Results window is also used to display the history list. To see the history list, click the History List tab at the bottom of the window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Back Button and the History List

InfoViewer keeps track of the topics you've previously viewed, so you can easily return to topics you've seen before.

{ewl msdncd, EWGraphic, jug14v 0 /a "build.bmp"} To return to the previous topic

```
{e Press this button on
w the Topic window
c toolbar. The default
m keyboard
sd shortcut is CTRL+B.
nc
d,
E
W
G
ra
p
hi
c,
ju
g
1
4v
1
/a
"u
g
ui
d
e.
B
M
P"
}
```

{ewl msdncd, EWGraphic, jug14v 2 /a "build.bmp"} To view a list of the topics you've previously visited

- From the View menu, choose InfoViewer History List.
The History List window appears.

Double-click the name of a topic to go there directly. You can keep the History list open while you view another topic by “pinning it”; click the push-pin button near the upper left corner.

By default, the History list displays the topics in the order you visited them, starting with the most recently visited.

{ewl msdncd, EWGraphic, jug14v 3 /a "build.bmp"} To sort the history list

- To sort the results list by book title, click the “Book” button at the top of the second column. To sort the results list by the topic title, click the “Topic” button at the top of the third column.

Note that the History List window is also used to display the results of your last query. To see the query results, click the Query Results tab at the bottom of the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Bookmarks

Bookmarks are flags that you can attach to a topic which enable you to return to the topic immediately. Bookmarks are persistent, so the next time you load the Microsoft Developer Studio, you can return to a useful topic without having to repeat a time-consuming browse or search operation. Bookmarks also retain the position within the topic, so you can find your original location in a lengthy topic without scrolling. You can even have multiple bookmarks within a single topic.

{ewl msdncd, EWGraphic, jug15v 0 /a "build.bmp"} To create a bookmark at the current topic

```
{e This toolbar button
w opens the Add
c Bookmark dialog
m box. If the toolbar
sd is not visible, click
nc the right mouse
d, button inside the
E Topic window and
W choose Add
G Bookmark from the
ra pop-up menu. The
p default keyboard
hi shortcut is
c, CTRL+SHIFT+B.
ju
g
1
5v
1
/a
"u
g
ui
d
e.
B
M
P"
}
```

The Add Bookmark [dialog box](#) proposes the current topic title as the default bookmark name. You can press the OK button if you want to accept the default bookmark name, or you can type a different name.

{ewl msdncd, EWGraphic, jug15v 2 /a "build.bmp"} To return to a bookmark

1 From the Edit menu, choose InfoViewer Bookmarks.

- 2 The InfoViewer Bookmarks dialog box appears. Select the name of the bookmark you want to return to and press the Display button. (You can also double-click the name of the bookmark to go there directly.)

An alternate method is to choose Go To from the Edit menu. This displays the Go To dialog box, from which you can jump to InfoViewer Bookmarks.

{ewl msdncd, EWGraphic, jug15v 3 /a "build.bmp"} To navigate through the bookmark list

```
{e Display the
w InfoViewer toolbar
c (from the View
m menu, choose
sd Toolbars; in the
nc Toolbars dialog
d box, select the
E InfoViewer check
W box). You can use
G these toolbar
ra buttons to move to
p the next or previous
hi topic in the results
c, list.
ju
g
1
5v
4
/a
"u
g
ui
d
e.
B
M
P"
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Customizing Books Online

[Back to the Introductory Screen](#)

[Customizing the Display](#)

[Using Subsets](#)

[Adding Annotations](#)

[Viewing Different Information Titles](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Customizing the Display

Click the right mouse button inside the Topic window to display the pop-up menu. You can do several things from this menu:

- To display or hide the toolbar or the topic title, select or clear Show Toolbar or Show Title, respectively. (Note that if you hide the Topic window's toolbar, you can still make annotations or add a bookmark using the pop-up menu or keyboard shortcuts.) The default keyboard shortcuts are CTRL+SHIFT+T and CTRL+T, respectively.
- To dock the Topic window, check Docking View. This allows you to dock the Topic window on any edge of the application frame. When the Topic window is docked, source windows will not overlap it, so you can always see both help and your source code at once.

You can also set other options by choosing Options from the pop-up menu. (You can also pull down the Tools menu and choose Options from there.) The Options dialog box opens to the InfoViewer tab. From this dialog box you can toggle the display of the toolbar or topic title, as well as set the following attributes:

- Underline jumps: This toggles the display of underlines on hot-links.
- Track topics in InfoView: This causes the Table of Contents to automatically expand nodes and scroll to remain synchronized with the currently open topic.
- Load information title at startup: This causes Microsoft Developer Studio to open the last information title at startup. You can turn this off if you're using Developer Studio without an information title and you don't want to be prompted for the title at startup.
- Zoom Topic title/Zoom Topic text: These let you specify the degree of enlargement used when displaying the title or topic text for an InfoViewer topic.
- Display highlights: When a topic is displayed following a full-text search, this toggles highlighting of search hits in topic text. See Full-Text Search.
- Jump to first highlight: When a topic is displayed following a full-text search, this causes the topic to be scrolled to display the first occurrence of the word or phrase that was searched for. See Full-Text Search.
- NEAR Means within X words: This lets you specify how close two words have to be to satisfy a query using the operator NEAR. See Operators: AND, OR, NOT, and NEAR.
- Context-Sensitive Help Subset: This lets you specify the subset you want to use for F1 help. See Using Subsets.

To customize the text colors for the Topic window, select the Format tab of the Options dialog box and select InfoViewer Topic Window in the Category scroll box. You can change the color of the topic text, text selections, highlights, jumps, or pop-ups.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Subsets

If you are interested in only a portion of the information available in the information title, you can instruct InfoViewer to use a “subset”—containing only the information you’re interested in—for operations such as browsing the Table of Contents, queries (full-text searches), or F1 help. To see the available subsets, choose Set Default Subsets from the Help menu. The Set Default Subsets dialog box displays the current subset for each operation, and lets you choose a different subset from a drop-down list. Notice that one of the subsets is named “Entire Contents,” which is the default subset. You can also define your own custom subsets. See Defining a New Subset.

When a subset is in effect for the Table of Contents, only nodes within that subset are displayed in the InfoView pane of the project workspace window. For more information, see Table of Contents. (If the InfoViewer Contents toolbar is visible, the subset currently in effect for the Table of Contents is displayed in the Select Subsets drop-down list.)

When a subset is in effect for queries (full-text searches), only the books within that subset are searched for the occurrences of the specified word or phrase. For more information, see Full-Text Search.

When a subset is in effect for F1 help, only that subset is searched for information on the word that you requested information on. For more information, see F1 Help.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Defining a New Subset

A subset lets you focus your searches or your browsing to topics in a particular category. For example, if you're interested only in the Java Application Programming Interface (API), Volume 1: Core Packages, you could define a subset that contains only the topics from the Java API documentation. For more on available subsets, see Using Subsets.

{ewl msdncd, EWGraphic, jug19v 0 /a "build.bmp"} To define a new subset

- 1 There are several ways to open the Define Subset dialog box. You can pull down the Help menu and choose Define Subset. You can also click the right mouse button in the Table of Contents window and choose Define Subset from the pop-up menu. You can also press the Subset button from the Query tab of the Search dialog box. If the InfoViewer Contents toolbar is visible, you can use the Define Subset button. The default keyboard shortcut from within an InfoViewer Topic window is CTRL+D.
- 2 The Define Subset dialog box appears. Type a name to identify your subset (or choose an existing subset from the drop-down list to edit that subset's contents).
- 3 Select the books you want for your new subset:
To add a book or book set to your selection, expand the list in the Available Books box, select it and then press the Add button.
To remove a book or book set from your selection, select its node from the Books in Subset box, and then press the Remove button.
- 4 Optional: select the Include New Topics Only check box if you want only the topics that are new since the last release of the documentation. (These are the topics that are marked with a red dot in the Table of Contents.) This option is available only if New Topics have been marked for the current information title as they are in the Microsoft Development Library.
- 5 Press the Save button, and then press Close.

When you first define a new subset, the subset automatically goes into effect in the context you created it. That is, if you define a new subset from the Help menu, the new subset appears in the Table of Contents listing. If you define a new subset from the Query tab of the Search dialog box, the new subset is used as the scope for full-text searching. Once you've saved a subset, you can use it in any context, not just the one in which you defined it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding Annotations

You can add your own notes or comments to any topic in the form of an annotation. You can view this annotation whenever you view that topic again, until you delete the annotation.

{ewl msdncd, EWGraphic, jug20v 0 /a "build.bmp"} To annotate the current topic

```
{e This toolbar button  
w opens an annotation  
c pane at the bottom  
m of the Topic window.  
s If the toolbar is not  
d visible, click the right  
n mouse button in the  
c Topic window and  
d choose Annotation  
E from the pop-up  
W menu. The default  
G keyboard  
ra shortcut is  
p CTRL+SHIFT+A.  
hi  
c,  
ju  
g  
2  
0  
v  
1  
/a  
"u  
g  
ui  
d  
e.  
B  
M  
P  
"} }
```

Type your notes in the annotation pane. You can resize the pane by dragging the bar above it.

To close the annotation pane, press the [toolbar](#) button again. Notice that the button changes color, indicating that an annotation exists. To view an annotation, press the same button. You can delete an annotation by deleting the text in the annotation pane.

To return to an existing annotation, choose Go To from the Edit menu. This displays the Go To dialog box, from which you can jump to InfoViewer Annotations.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Viewing Different Information Titles

In InfoViewer, a help file is known as an information title and has a .MVB file extension.

{ewl msdncd, EWGraphic, jug21v 0 /a "build.bmp"} To open a different information title that has been loaded into InfoViewer

- 1 While an InfoViewer Topic window has focus, open the Open Information Title dialog box by pressing CTRL+SHIFT+O (this is the default shortcut). Alternately, if the InfoViewer toolbar is visible, you can use the Open Information Title drop-down list.

The Open Information Title dialog box appears.

- 2 Select the name of the information title you want to view.

The InfoView pane of the Project workspace now displays the Table of Contents for the new information title. If the InfoViewer toolbar is visible, it displays the name of the new information title.

You can also open an information title via the File Open dialog box. Specify the title's .MVB name in the File Name box.

For each information title, InfoViewer maintains a separate history list, bookmark list, list of subsets, and last query results.

If you use other products that use the Microsoft Developer Studio as the development environment, you may need to load the Visual J++ information title (VJBKS10.MVB) into InfoViewer the first time you open a Visual J++ project workspace or want information about a Java specific topic. If Visual J++ Books Online 1.0 is not an item in the drop-down list box on the InfoViewer toolbar, you'll need to tell InfoViewer where to find the title.

{ewl msdncd, EWGraphic, jug21v 1 /a "build.bmp"} To open an information title that has not been loaded into InfoViewer

- 1 While an InfoViewer Topic window has focus, select Open from the File menu or press CTRL+O (this is the default shortcut). Alternately, if the File toolbar is visible, you can click the Open button.

The Open dialog box appears.

- 2 Navigate to the directory that contains the VJBKS10.MVB file. (C:\MSDEV\HELP is the default installation directory—check directory path and name.)
- 3 Select All Files (*.*) from the Files of Type drop-down list box.
- 4 Select the Visual J++ information title, VJBKS10.MVB, or type the .MVB file name into the File Name edit box.
- 5 Click the Open button.

The InfoView pane of the Project workspace now displays the Table of Contents for Visual J++'s information title. If the InfoViewer toolbar is visible, it displays the name of the new information title.

Use this same process to load the .MVB files for other products and tools that work within Microsoft Developer Studio.

For each information title, InfoViewer maintains a separate history list, bookmark list, list of subsets, and last query results.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Special Features

[Back to Introductory Screen](#)

[Sample Programs](#)

[Copying and Printing](#)

[Toolbars](#)

[Keyboard Shortcuts](#)

[Customizing Keyboard/Toolbars](#)

[Tips for Using InfoViewer](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Sample Programs

InfoViewer provides access to all sample applications. These source files are accessible from the sample abstracts, which are topics describing what each sample application does. For each sample, you can view the individual source files, run the program, or copy the source files onto your hard disk to build the application yourself.

The sample abstracts are found in the Table of Contents under the node named Samples, and are grouped by category. Each sample abstract contains the following button:

```
{ewc msdncd, EWGraphic, jug23v 0 /a "uguide.BMP"}
```

Press this button to open the Sample Application dialog box. This dialog box displays a listing of all the files associated with the sample application. If you want to look at a particular file, select the file in the file list and press the View button. If you want to build the program, copy the files to your hard drive, and create a Java project. See Building a Sample Java Applet for more information.

When you extract the files onto your hard drive, select the desired files and press the Copy button, or press the Copy All button to copy the entire sample. Both the Copy and Copy All buttons let you specify a target directory for the files being copied.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Copying and Printing

You can copy text in the Topic window for use in your application. You can also print a topic or a series of topics, or print the the Table of Contents.

{ewl msdncd, EWGraphic, jug24v 0 /a "build.bmp"} To copy text from a Topic window

- 1 Select the text you want to copy. (You can also select the entire topic by clicking the right mouse button within the Topic window and choosing Select All from the pop-up menu.)
- 2 Drag the text to the desired location. (You can also copy it to the Clipboard by clicking the right mouse button and choosing Copy. In addition to plain text, InfoViewer also places Rich Text Format (RTF) on the Clipboard so that formatting information is retained if you paste the text into an application that accepts RTF.)

{ewl msdncd, EWGraphic, jug24v 1 /a "build.bmp"} To print a single topic

- Click the right mouse button inside the Topic window and choose Print from the pop-up menu. The default keyboard shortcut is CTRL+P. To view the Print dialog box (which lets you selectively print to a file, print annotations, and so on), pull down the File menu when the Topic window has FOCUS and choose Print.

{ewl msdncd, EWGraphic, jug24v 2 /a "build.bmp"} To print multiple topics

- 1 Open the Table of Contents by choosing the InfoView tab at the bottom of the Workspace window.
- 2 Select the topics you want to print. To print an entire book, simply select the book-level node. To select a continuous range of topics within a book, expand the book and—while holding down the SHIFT key—use the cursor keys to select the topics you want. To select non-adjacent topics, hold down the CTRL key and either click the mouse on the desired topics, or use the ARROW keys and the spacebar.

Note If you select a book-level node for printing, you will print all the topics contained within that book. Consequently, if a book is expanded and you select both the book-level node and some page-level topics contained within it, you will print the individual topics twice.

- 3 Click the right mouse button inside the selection to display the pop-up menu, and choose Print. (You can also choose Print from the File menu.) In the Print dialog box, choose the Selected Topics Or Books option button and press OK.

{ewl msdncd, EWGraphic, jug24v 3 /a "build.bmp"} To print the Table of Contents

- Click the right mouse button inside the Table of Contents to display the pop-up menu, and choose Print. (You can also choose Print from the File menu.) In the Print dialog box, choose the Content Listing button and press OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolbars

InfoViewer provides two floating toolbars; these toolbar buttons offer one-click execution for a number of help commands. These toolbars can be docked on any edge of the application frame, or they can float as miniframe child windows.

To display (or hide) these toolbars, pull down the View menu and choose Toolbars. In the Toolbars dialog box, select (or clear) the InfoViewer or InfoViewer Contents check boxes.

The InfoViewer Toolbar

```
{ewc msdncd, EWGraphic, jug25v 0 /a "uguide.BMP"}
```

```
{ewc msdncd, EWGraphic, jug25v 0 /a "uguide.BMP"} This button returns you to the home screen of the information title.
```

```
EWGraphic, jug25v 0 /a "uguide.BMP"} This control lets you choose the information title you want to view.
```

```
- This button opens the Search dialog box.
```

```
EWGraphic, This button opens the Search dialog box.
```


jug
25v
2 /a
"ug
uid
e.B
MP
"}
{ew

c
ms
dnc
d,
EW
Gra
phi
c,
jug
25v
3 /a
"ug
uid
e.B
MP
"}
{ew

c
ms
dnc
d,
EW
Gra
phi
c,
jug
25v
3 /a
"ug
uid
e.B
MP
"}
{ew

c
ms
dnc
d,
EW
Gra
phi
c,
jug
25v
4 /a
"ug
uid
e.B
MP
"}
{ew

c
ms
dnc
d,
EW
Gra
phi
c,
jug
25v
4 /a
"ug
uid
e.B
MP
"}
{ew

c
ms
dnc
dnc

This button
displays the Query
Results window.

These buttons let
you navigate the
Query Results list.

This button
displays the
InfoViewer
Bookmarks [dialog](#)

d, [box](#).

EW

Gra

phi

c,

jug

25v

5 /a

"ug

uid

e.B

MP

"}

{ew These buttons let
c you navigate the
ms list of bookmarks.

dnc

d,

EW

Gra

phi

c,

jug

25v

6 /a

"ug

uid

e.B

MP

"}

The InfoViewer Contents Toolbar

{ewc msdncd, EWGraphic, jug25v 7 /a "uguide.BMP"}

{ew This button
c displays the
ms Define Subset
dnc [dialog box](#).

d,

EW

Gra

phi

c,

jug

25v

8 /a

"ug
uid
e.B
MP
"}
Set This control lets
Defa you choose the
ult subset you want
Sub to [view](#).
set
drop
-
dow
n list

For information on moving toolbar buttons around, see [Customizing Keyboard/Toolbars](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Keyboard Shortcuts

If you prefer the keyboard to the mouse, you can use the [shortcut keys](#) listed below. The shortcuts listed are the default for the Developer Studio keyboard mapping; if you chose another keyboard mapping on installation, you'll have a different set of [shortcut keys](#). For information on reassigning these commands to different keys, see [Customizing Keyboard/Toolbars](#).

To see a list of all current keyboard assignments, pull down the Help menu and choose Keyboard.

Working with [panes](#)

To	Press
Show/Hide Topic Title Bar	CTRL+T
Show/Hide Topic window toolbar	CTRL+SHIFT+T
Open or close the Annotation pane	CTRL+SHIFT+T+A
Add a Bookmark	CTRL+SHIFT+T+B

Navigating in the Contents pane

To	Press
Open current book one level in the Contents pane	RIGHT ARROW or PLUS SIGN
Close current book one level in the Contents pane	LEFT ARROW or MINUS SIGN

Displaying topics

To	Press
Display the last	CTRL+B

viewed topic	
Display the previous topic in the Contents list	CTRL+SHIF T+P
Display the next topic in the Contents list	CTRL+SHIF T+N
Display See Also list	CTRL+SHIF T+S
Synchron ize Table of Contents list	CTRL+S

Searching

To	Press
Open the Search dialog box	CTRL+F
Highlight search terms in topics (toggle)	CTRL+H
Open the Define Subset dialog box	CTRL+D

Miscellaneous

To	Press
Display the home screen	CTRL+E
Print entire topic	CTRL+P
Open	CTRL+SHIF

another T+O
help title

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Customizing the Keyboard and Toolbars

InfoViewer provides two floating toolbars, described under Toolbars, and defines a set of default keyboard shortcuts, listed under Keyboard Shortcuts. However, you can assign help commands to different keys or design a custom toolbar that contains only the buttons you want. To see a list of all current keyboard assignments, pull down the Help menu and choose Keyboard.

{ewl msdncd, EWGraphic, jug27v 0 /a "build.bmp"} To customize your keyboard

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Editor list box, select the editor whose key assignments you want to change. By default, help shortcuts are defined only within the InfoViewer editor, that is, when an InfoViewer Topic window has focus. If you define a keyboard shortcut for the Main editor, the shortcut will work no matter which editor you're using.
- 4 In the Categories list box, select Help.
The Commands list box displays all the predefined macros that are related to help. When you highlight an individual macro, the Description box describes what it does, and the Current Keys box displays the current key assignment(s), if any.
- 5 Select a macro and press the new shortcut key you want the macro assigned to.

{ewl msdncd, EWGraphic, jug27v 1 /a "build.bmp"} To customize your toolbars

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Toolbars tab.
- 3 In the Categories list box, select Help.
The Buttons box displays all the predefined toolbar buttons that are related to help. When you select an individual button, the Description box describes what it does.
- 4 Select a button and drag it onto a toolbar currently visible in the workspace.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Tips for Using InfoViewer

- You can drag and drop text from the InfoViewer Topic window to a source window.
- You can toggle the display of the InfoViewer Topic title by pressing CTRL+T. You can toggle the InfoViewer Topic toolbar by pressing CTRL+SHIFT+T.
- You can define a subset to display a smaller set of books in the InfoView pane, to narrow the scope of a query, or to narrow the scope of an F1 search in a source file. See Using Subsets.
- TAB and CTRL+J move to the next hot-link in the InfoViewer Topic window.
- Dock your InfoViewer toolbar to either side of the application frame for easy access.
- In an InfoViewer topic, when you click jump text, you go to a related topic. To return to the original topic, press CTRL+B.
- Pin the InfoViewer Query Results window to browse and display topics. When the window is unpinned, it disappears when you open a topic. Choose InfoViewer Query Results from the View menu to display it again. See Browsing Query Results.
- In a source file, F1 displays information about the word at the insertion point. Using F1 works in the InfoViewer Topic and Output windows, too! When you select text, F1 displays information about the selected text. See F1 Help.
- You can query for a search string in text or just in titles. Querying for a search string just in titles is a good technique to find a topic you have visited before.
- If there is more than one relevant topic, F1 keyword help in a source file will display a list of topics for the current subset in the Select Reference dialog box. To narrow your search, use a subset specific to your programming. To broaden your search, use a subset with a greater range of books. You can change your current subset from the Select Reference dialog box or by using the Set Default Subsets command on the Help menu. See Using Subsets.
- There are two default InfoViewer toolbars: InfoViewer, which contains buttons for common operations, and InfoViewer Contents, which lists available subsets. See Toolbars.
- Choose Keyboard on the Help menu to see the current keyboard shortcuts. The keyboard list is updated when you make a change to a shortcut assignment. Change assignments from the Customize command on the Tools menu. See Keyboard Shortcuts.
- You can dock the InfoViewer Topic window. Choose Docking View from the pop-up menu to switch the topic window into a floating state, then drag the topic window to the application frame to dock it.
- In the InfoViewer Topic window, CTRL+SHIFT+S displays the See Also list. It works even if you've hidden the toolbar with CTRL+SHIFT+T.
- You can dock the InfoViewer Contents toolbar at the top of the Workspace window by pressing SHIFT while dragging the InfoViewer Contents toolbar into position. The InfoViewer Contents toolbar allows you to quickly change subsets.
- In an InfoViewer topic, when you click pop-up text (dark gray color by default), you see the definition of the term.
- You can change the color of the glossary pop-up text to any color. Use the Format tab in the Options dialog box to change pop-up text color.
- You can turn off title tips in the InfoView pane by toggling Title Tips using the pop-up menu.
- You can open Microsoft Development Library from within Microsoft Developer Studio. Press CTRL+SHIFT+O to open the Open Information Title dialog box. See Viewing Different

Information Titles.

- You can use Microsoft Development Library for F1 help in source files rather than Visual J++ Books Online. If you are a subscriber to MSDN, just open the latest version from the Open Information Title dialog box. Until you open another title, it will remain current between sessions.
- Files in online sample programs are opened up directly in Developer Studio. See the Samples node in Books Online or Microsoft Development Library. See Sample Programs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Projects

The project workspace organizes your projects and their elements, and maintains your preferences for the display of information. The project workspace consists of a subdirectory and various files. The files describe the individual projects in the project workspace, and how to display them.

There are two basic scenarios for using project workspaces. Before you create your project workspace, you should determine which scenario for project workspace organization suits your needs best. You can modify these workspace scenarios in a number of ways to fit your specific requirements.

When you create or open a project workspace, Microsoft Developer Studio displays the elements of your Project Workspace in the project workspace window, as shown in Figure 4.1.

Figure 4.1 The Project Workspace Window and Project Toolbar

{ewc msdncd, EWGraphic, jug0b 0 /a "uguideWKSPW.BMP"}

When you open a project workspace file, Developer Studio displays the Project Workspace window, along with other windows, in the last locations and states that you chose for them. You can dock or undock, size, move, or hide the Project Workspace window.

In the Project Workspace window, Developer Studio creates panes you access from the tabs at the bottom of the window. Certain panes contain a specific view of all the projects in your workspace. Each pane has at least one top-level folder that contains the elements that make up that view of the project; expanding the folder displays the details of that view. In a project workspace containing Visual J++ projects, for instance, the Project Workspace window contains the following panes by default.

Pan e Title	Description
<u>File View</u>	Displays the projects that you have created. Expanding the top-level folders shows files within the project.
<u>Class View</u>	Displays the Java classes defined in your projects. Expanding the top-level folders shows the classes; expanding a class shows its

members.

Info
Vie
w Displays the
table of
contents for
Books Online.
Expanding the
top-level
folders shows
books and
topics.

Each folder within a view can contain other folders or various kinds of items. The items may consist of subprojects, classes, topics, and other project files.

It is important to keep in mind that the organization of the items in a folder represents the relationships of the items in the project, not the physical location of items. A project folder, for instance, contains icons representing the files used to build the project. The icons show whether or not the files are used in the build process, as well as the relationship of source files to their dependent files. Those files could reside in any directory on any drive accessible from your machine.

Note Do not confuse a Java package with a Developer Studio subproject. The environment finds classes in a Java package using the classpath information. If you move a Java package to another location on your drive, update the classpath information to reflect the new location of the package. See CLASSPATH environment variable for more information.

Initially, the default project configuration is shown in Project toolbar's drop-down list box. If you choose a build command, you build that default project configuration.

You can access information about elements of the project workspace from the views in the Project Workspace window. Selecting any item and pressing ALT+ENTER opens the property page for that item. Double-clicking any item in a pane displays that item in an appropriate way: source files in a text editor, dialog boxes in the dialog editor, information topics in the topic window, and so on.

Pop-up menus take action on selections in the Project Workspace window. When you make a selection, and then press the right mouse button with the mouse pointer over the selection, the pop-up menu appears. It contains commands appropriate for the selection.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Project Workspaces

In Microsoft Developer Studio, you organize your work in a project workspace. A project workspace consists of a location—the workspace directory—and some files in that directory, which describe the workspace and its contents. When you first create a project workspace, you create a directory for the project workspace and a project workspace file with the extension .MDP. The project workspace file is what you save when you have completed working in your new project workspace and what you open when you want to resume work in your project workspace.

The project workspace directory is the root directory for the project workspace, and all packages and subsequent projects that you add to this project workspace are added in subdirectories under the project workspace directory. By default, Developer Studio selects the Projects subdirectory under the Developer Studio installation directory as the initial location for all your project workspace directories. You can, however, choose another location. If you do choose another location, Developer Studio retains that location as the initial location for subsequent project workspaces that you create. The project workspace directory contains the following files:

- The project workspace file (.MDP)
- The project workspace makefile (.MAK)

Usually all source files associated with the project are created in the project workspace directory. You can add source files to the project from any location, however, without copying or moving them to this subdirectory.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Elements of Project Workspaces

Project workspaces have the following elements:

Project A set of zero or more source files, with one or more configurations. A project also specifies the type of program to build. Your project workspace can contain any number of projects. It can contain subprojects.

Configuration Settings for a project that specify a platform on which the output file is to run, and tool settings with which to build the output. You can add any number of configurations to a project. By default, when you create a new project, you create Debug and Release configurations.

Within a project workspace, projects can have the following relations:

Java project A project that is not a dependency of any other project. Not a subproject of any other

project. A Java project [workspace](#) has at least one Java project.

[Sub project](#) A subproject in a Java [workspace](#) does not have a dependency relationship with another project. One of the main benefits of using a subproject within Visual J++ is that it gives you the ability to add projects written in [native](#) code. Since there is no dependency relationship, the [build](#) system does not determine if it needs to [build](#) the subproject before it builds the containing project. To make sure you have incorporated the latest changes from all projects and subprojects, choose Rebuild All from the Build menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Files Associated with Project Workspaces

When you create a project workspace, Microsoft Developer Studio creates two or more associated files. In the case of Visual J++ projects, for instance, it creates a project makefile (.MAK) and a project workspace file (.MDP) to store information.

The .MAK file stores the following kinds of information required to build the project:

- The names and locations of the source files that are used to build each project.
- The settings for the tools required to build each project, such as compiler options.
- The tools and actions required to build the project.

The .MDP file stores the following kinds of information for your particular workspace:

- The look and organization of Developer Studio for the project workspace (choice and locations of windows, for instance).
- Breakpoints that you have set.
- Other information related to your local setup, such as fonts and colors.

If you work in a group, you generally want to share the makefile with other members of your group, so that they can build the projects defined in the project workspace. To do this, see Maintaining Makefiles Under Source-Code Control for more information. You probably should not share the project workspace file, because it contains information about your local organization and appearance.

Developer Studio may also generate a number of other files, depending on the project type and the settings that you choose. Developer Studio manages these files for you.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Project Workspaces: Two Basic Scenarios

There are two basic scenarios for using Visual J++ project workspaces. These consist of the following cases:

- A Java project only
- A top-level project with a single subproject

These are very basic, general organizations. You can modify or expand them in ways to serve your particular development needs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating a Java Project

This organization is suitable for the development of a single program. Choose this organization if you want to develop, for instance, a single applet generated by Applet Wizard, or a stand alone Java application. Figure 4.2 shows the relationships among the elements of a Java project.

Figure 4.2 Java project

{ewc msdncd, EWGraphic, jug5b 0 /a "uguideSIMPL.BMP"}

{ewl msdncd, EWGraphic, jug5b 1 /a "build.bmp"} To create a Java project

- 1 From the File menu, choose New.
The New dialog box appears.
- 2 From the New list, choose Project Workspace.
The New Project Workspace dialog box appears.
- 3 Select the project type from the list of types.

Note The Java Workspace creates an empty project; the Java Applet Wizard generates project files based upon options you select.

- 4 The Platforms list box should display Java Virtual Machine as the platform for your Java program.
- 5 In the Name text box, type a name for the project workspace.
This name is also the name of the initial Java project.
- 6 If you do not want to use the default directory, PROJECTS, in the Location text box, type another directory name in which you want to create this project workspace subdirectory .
If you revise this location, Developer Studio retains the new location as the default for creating new project workspaces.
- 7 Choose Create.

Now you can add files if necessary, modify the source code, add methods to classes, add variables to classes, or add a new class to the project, change the project configuration settings for your program, and so on.

Note If you chose to create a Java Applet Wizard project, the wizard creates a set of starter files for your program. You can now modify those files to complete your Java applet or stand-alone Java application.

You build a configuration of your program by selecting a configuration to build, using the Default Configuration drop-down list on the Project toolbar, and then choosing the Build command from the Build menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Top-level Project with a Single Subproject

A top-level project with a single subproject is an organization that has a Java project with a single native code subproject or a native code top-level project with a Java subproject. You could choose this organization if you want to develop, for instance, a stand-alone Java application that uses a C++ subproject or vice versa. Figure 4.3 shows the relationships among the elements of a Java project with a single subproject.

Note Do not confuse a Java package with a Developer Studio subproject. When a Java program runs, the environment finds classes in a Java package using the classpath information. If you move a Java package to another location on your drive, update the classpath to reflect the new location of the package. See CLASSPATH environment variable and Setting Directories for more information.

Figure 4.3 Top-level project with a Single Subproject

```
{ewc msdncd, EWGraphic, jug6b 0 /a "uguideSIMSUB.BMP"}
```

{ewl msdncd, EWGraphic, jug6b 1 /a "build.bmp"} To create a top-level project with a subproject from the Build menu

- 1 Create a new project.
- 2 From the Build menu, choose Subprojects.
The Subprojects dialog box appears.
- 3 Choose New.
The Insert Project dialog box appears, with the Subproject option selected, and the existing Java project selected in the drop-down list. Retain these default choices.
- 4 In the Name text box, type a name for the subproject.
This name is appended to the existing project workspace directory to form the fully qualified path for the new subproject directory.
- 5 From the Type list, select a project type.
- 6 Select any of the available platforms for which to create initial Debug and Release configurations.
- 7 Choose Create.
After you have responded to any wizard dialog boxes, the Subprojects dialog box reappears. Your newly created subproject is selected for inclusion.
- 8 Choose Close.

{ewl msdncd, EWGraphic, jug6b 2 /a "build.bmp"} To create a top-level project with a subproject from the Insert menu

- 1 Create a new project.
- 2 From the Insert menu, choose Project.
The Insert Project dialog box appears.
- 3 Click the Subproject of radio button
The name of the Java project displays in the drop-down list.
- 4 In the Name text box, type a name for the subproject.
This name is appended to the existing project workspace directory to form the fully qualified

path for the new subproject directory.

- 5 From the Type list, select a project type.
- 6 Select any of the available platforms for which to create initial Debug and Release configurations.
- 7 Choose Create.

After you have responded to any wizard dialog boxes, the Subprojects dialog box reappears. Your newly created subproject is selected for inclusion.

- 8 Choose Close.

You have completed creating a Java project with a single subproject. In FileView, the Java project icon, when expanded, displays an icon representing the dependency relation for the subproject. The subproject also has a top-level representation.

Now you can add files if necessary, modify the source code, add methods to classes, add variables to classes, or add a new class to the project, change the settings for your program, and so on.

When you build in this project workspace, you can build either the subproject or both the Java project and the subproject.

{ewl msdncd, EWGraphic, jug6b 3 /a "build.bmp"} To build the subproject only

- 1 Select the subproject with the configuration you want to build from the Set Default Project Configuration drop-down list on the Project toolbar.
–or–
From the Build menu, choose Set Default Configuration, and choose from the list in the Default Project Configuration dialog box.
–or–
Select Set Default Project from the right mouse pop-up menu.
- 2 From the Build menu, choose Rebuild All.

{ewl msdncd, EWGraphic, jug6b 4 /a "build.bmp"} To build both the Java project and the subproject

- 1 Select the Java project with the configuration you want to build from the Set Default Project Configuration drop-down list on the Project toolbar.
–or–
From the Build menu, choose Set Default Configuration, and choose from the list in the Default Project Configuration dialog box.
- 2 From the Build menu, choose Build.

If the subproject output file is out of date, the build system first builds it, and then it builds the Java project. If it is not out of date, the build system builds only the Java project.

If you want to force the system to rebuild all the output files, choose Rebuild All from the Build menu. This method ensures, for instance, that you are always testing both elements of your project workspace with the most up-to-date changes.

Note If you have an output .CLASS file created by a project which calls a class created by a subproject, you need to do one of three things in order to run the program and have it find the class file. You can add the output directory for the class file to your classpath, you can specify the output directory for the class file to be same as the output directory for the program, or you can

move the class file after it is built to a directory on the classpath, using a custom build command. In the case when you're debugging, it is preferable to set the output directories to be the same for the Java program and the class file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Managing Project Workspaces

Project workspaces contain projects that you can build. A project consists of a single set of files and a set of one or more project configurations. Each project configuration, together with the set of files, determines the .CLASS output file that you create.

When you create a project workspace, by default you always create one project with two configurations for each platform:

- A version with debugging information included (Debug)
- A version with no debugging information (Release)

After you have created that initial project workspace, you can add:

- New projects to your existing workspace.
- New configurations to an existing project.
- Subprojects to any project.

A subproject establishes a dependency of one project on another. A project that builds a Java program that depends on another class file is one example. If the class file is a subproject of the project that builds the program, then the class file will be updated before the program is built. Each configuration of a subproject is made a dependency of the corresponding configuration in the containing project. Building a configuration of the program also builds the same configuration of the subproject.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating a Project Workspace

When you start a software development task with Microsoft Developer Studio, you create a project workspace and an initial project in the workspace. The initial project has Debug and Release configurations for each platform that you choose. Before you create your project workspace, you should determine which of the basic scenarios for project workspace organization suits your needs best.

When you create a project workspace, you select a root directory in which to create your project workspace directory. By default, Developer Studio selects the PROJECTS directory under your installation directory. You can, however, choose another directory. If you choose another directory, Developer Studio uses that choice for all subsequent project workspaces that you create.

When you create a project workspace, you must specify a name that is used both for the project workspace directory and the initial project in the workspace. Developer Studio creates a subdirectory of this name in the root directory. This subdirectory contains the files for your project workspace and the files for your initial project.

Developer Studio also specifies subdirectories for final output files for the various projects that you specify. These subdirectories enable you to build various configurations of a project without overwriting final output files with the same names. With the Settings command on the Build menu, you can open the General tab in the Project Settings dialog box and modify these subdirectories.

When you create the initial project in a new project workspace, you automatically create two configurations: Debug and Release. The Debug version specifies settings to include debugging information. The Release version doesn't specify settings to include debug information.

Note If you add files from directories above the project workspace directory, Developer Studio uses relative paths in the filenames for those files in the project's .MAK file. Because of these paths, it is may be difficult to share the .MAK file. Other developers in your group may have other drive names or higher-level directory structures. See Maintaining Makefiles Under Source-Code Control for further information about sharing makefiles.

{ewl msdn cd, EWGraphic, jug8b 0 /a "build.bmp"} To create a project workspace

- 1 From the File menu, choose New.
The New dialog box appears.
- 2 Select Project Workspace from the list.
- 3 Choose OK.
The New Project Workspace dialog box appears.
- 4 From the Type list, select the type of program that you want to create.
- 5 In the Name text box, type the name for the project workspace. This name is also used for the initial project in the Project Workspace window.
Developer Studio automatically creates a new subdirectory with this name for your project workspace and for the files for the initial project.
- 6 Select the platform type or types from the Platforms list.
If you have chosen a Java Workspace or Java Applet Wizard project type, you will not have a choice of platform types. When your project builds, it will run on the Java Virtual Machine (VM)

which is just another type of platform.

7 If you want to change to location of your project, type a new location for the root directory in the Location text box, or choose Browse and select another location.

8 Choose Create.

If you chose a Java Workspace project, it will not generate starter files. You will need to add files to your project. See [Adding and Removing Files from Projects](#) for more information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Choosing a Project Workspace Name

Microsoft Developer Studio accepts Project Workspace names that meet the following criteria:

- Must contain alphanumeric characters.
- May not begin with a dollar sign (\$).
- May contain spaces.
- A maximum of 128 characters excluding a project's path.
- If a path is specified, the path, including the project name, may not exceed MAXPATH - 32. In Win32 MAXPATH = 260, making the maximum number of characters allowed for a project name on a Win32 operating system 228.

Note The Java language specification states that a valid Java *identifier*, used for project, label, constant, and variable names may be of any length and may contain digits or letters or a combination of both. A valid identifier must also begin with a letter, a dollar sign (\$), or an underscore (_). Visual J++ supports this naming convention with the exception of the project name which must follow the criteria listed above.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Project Types

Each Visual J++ project has a project type, Java Workspace or Java Applet Wizard, which you determine when you create the project. The project type specifies what is generated and specifies some default settings required in order to build that output project type. It specifies, for instance, the settings that the compiler uses for the source files, the default locations for output files, defined constants, and so on.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Platform Types

The platform type for a project configuration specifies the operating environment. For Visual J++ this platform is the Java Virtual Machine, also called the Java VM. The platform type specifies default settings required by a given platform, such as settings that the compiler uses for the source files, the default locations for output files, defined constants, and so on. It also specifies the tools required to build the final output files for that platform.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Saving a Project Workspace

You can save the workspace files and all other files that you have modified with the Save All command.

{ewl msdncd, EWGraphic, jug12b 0 /a "build.bmp"} To save all files in a project workspace

- From the File menu, choose Save All.
Microsoft Developer Studio saves all files that you have modified—whether or not they are included in a project—without any further action on your part.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Closing a Project Workspace

You can close the workspace files with the Close Workspace command.

{ewl msdncd, EWGraphic, jug13b 0 /a "build.bmp"} To close a project workspace

- From the File menu, choose Close Workspace.
If necessary, Microsoft Developer Studio prompts for actions concerning the windows that are open and the files that you have modified—whether or not they are included in a project.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Opening an Existing Project Workspace

Opening an existing project workspace loads all project workspace information, and restores all the environment settings to their state when you last saved the project workspace.

{ewl msdncd, EWGraphic, jug14b 0 /a "build.bmp"} To open an existing project workspace

- 1 From the File menu, choose Open Workspace.
The Open Project Workspace dialog box appears.
The default selection in the List Files Of Type drop-down list is Project Workspaces (.MDP).
- 2 Select the drive and directory containing the project workspace that you want to open.
- 3 Select the .MDP file for the project workspace from the File Name list and choose OK.
—or—
Double-click the filename in the list.

The Project Workspace window appears, as shown in Figure 4.1, and displays views of the projects in the workspace.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Opening Other File Types

You can open file types other than project workspace files in the Project Workspace window. In particular, you can open makefiles (.MAK), or you can open program files to debug them.

{ewl msdncd, EWGraphic, jug15b 0 /a "build.bmp"} To open an existing makefile with the extension .MAK

- 1 From the File menu, choose Open Workspace.

The Open Project Workspace dialog box appears.

- 2 Select Makefiles from the drop-down list to display .MAK files.
- 3 Select the drive and directory containing the makefile that you want to open.
- 4 Select the .MAK file from the list and choose OK.

—or—

Double-click the filename in the list.

If you have a project workspace currently open, Developer Studio saves the workspace and asks if you want to close document windows associated with that workspace.

If your makefile has a different extension, or has the name MAKEFILE, you can use the Open command on the File menu to open it as a makefile.

{ewl msdncd, EWGraphic, jug15b 1 /a "build.bmp"} To open an existing makefile without the extension .MAK

- 1 From the File menu, choose Open.

The Open dialog box appears.

- 2 Select All Files from the drop-down list to display all files.
- 3 From the Open As drop-down list, select Makefile.
- 4 Select the drive and directory containing the makefile that you want to open.
- 5 Select the file from the list and choose OK.

—or—

Double-click the filename in the list.

If you have a project workspace currently open, Developer Studio saves the workspace and asks if you want to close document windows associated with that workspace.

The Project Workspace window appears, and Developer Studio takes the appropriate action for the file opened.

You can also open an .CLASS file and create a project workspace for it in order to debug it.

{ewl msdncd, EWGraphic, jug15b 2 /a "build.bmp"} To open a .CLASS file for debugging

- 1 From the File menu, choose Open Workspace.

The Open Project Workspace dialog box appears.

- 2 Select Java Class from the Files of Type drop-down list box.
- 3 Select the drive and directory containing the .CLASS file that you want to open.
- 4 Select the .CLASS file and choose OK.

—or—

Double-click the filename in the list.

If you have a project workspace currently open, Developer Studio saves the workspace and asks if you want to close document windows associated with that workspace.

The Project Workspace window opens and displays the .CLASS file as a folder in the FileView pane. You can now choose debugging commands from the Build menu, and debug the program. When you close the project workspace, Developer Studio asks if you want to save the new project workspace associated with this .CLASS file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Specifying Subprojects in a Project Workspace

Subprojects indicate relationships in the project workspace. The relationship is established by configuration. That is, if you build the Debug configuration of the containing project, you also build the Debug configuration of the subproject. Subprojects can contain other subprojects. All subprojects also have a top-level representation in the FileView pane.

When you specify a subproject, you can either create a new project and give it a subproject relationship, or you can choose an existing project and give it a subproject relationship.

{ewl msdncd, EWGraphic, jug16b 0 /a "build.bmp"} To create a new project as a subproject from the Build menu

- 1 From the Build menu, choose Subprojects.
The Subprojects dialog box appears.
- 2 From the Select Project To Modify drop-down list box, select the project that is to contain the new subproject.
- 3 Choose New.
The Insert Project dialog box appears, with the Subproject option selected, and the project displayed in the Subproject Of drop-down list box. Retain these default choices.
- 4 In the Name text box, type a name for the subproject.
This name is appended to the existing project workspace directory to form the fully qualified path for the new subproject directory.
- 5 From the Type list box, select a project type.
- 6 Select any of the available platforms for which to create initial Debug and Release configurations.
- 7 Choose Create.
After you have responded to any wizard dialog boxes, the Subprojects dialog box reappears. Your newly created subproject is selected for inclusion.
- 8 Choose OK.

{ewl msdncd, EWGraphic, jug16b 1 /a "build.bmp"} To create a new project as a subproject from the Insert menu

- 1 From the Insert menu, choose Projects.
The Insert Project dialog box appears.
- 2 Click the Subproject radio button. The current project's name displays in the Subproject Of drop-down list.
- 3 In the Name text box, type a name for the subproject.
This name is appended to the existing project workspace directory to form the fully qualified path for the new subproject directory.
- 4 From the Type list box, select a project type.
- 5 Select any of the available platforms for which to create initial Debug and Release configurations.
- 6 Choose Create.
After you have responded to any wizard dialog boxes, the Subprojects dialog box reappears. Your newly created subproject is selected for inclusion.
- 7 Choose OK.

{ewl msdncd, EWGraphic, jug16b 2 /a "build.bmp"} To include an existing project as a subproject

- 1 From the Build menu, choose Subprojects.
The Subprojects dialog box appears.
- 2 From the Select Project To Modify drop-down list box, select the project that is to contain the subproject.
The Select Subprojects To Include list box displays the projects that you can include as subprojects of the selected project. Projects that are already subprojects for this project have a check mark next to them.
- 3 From the Select Subprojects To Include list box, select the project (or projects) that you want to include as a subproject.
- 4 Choose OK.

If you have included a project in another project as a subproject, you can remove it from the project, and by doing so, remove its relationship. This does not, however, remove the project from the project workspace.

{ewl msdncd, EWGraphic, jug16b 3 /a "build.bmp"} To remove a subproject from a project

- 1 From the Build menu, choose Subprojects.
The Subprojects dialog box appears.
- 2 From the Select Project To Modify drop-down list box, select the project that now contains the subproject.
The Select Subprojects To Include list displays projects that are already subprojects with a check mark next to them.
- 3 From the Select Subprojects To Include list, select the project (or projects) that you want to remove as a subproject.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Views

In the Project Workspace window, Developer Studio creates panes, which you access from the tabs at the bottom of the window. Certain panes contain a specific view of all the projects in your workspace. Each pane has at least one top-level folder, which contains the elements that make up that view of the project; expanding the folder displays the details of that view. In a project workspace containing Visual J++ projects, for instance, the Project Workspace window contains the following panes by default.

Pan e Title	Description
File View	Displays the projects that you have created. Expanding the top-level folders shows files within the project.
Classes View	Displays the Java classes defined in your projects. Expanding the top-level folders shows the classes; expanding a class shows its members.
Info View	Displays the table of contents for Books Online. Expanding the top-level folders shows books and topics.

You can switch from one pane to another by selecting a tab at the bottom of the Project Workspace window, as shown in Figure 4.4. You can also switch using CTRL+PAGE UP and CTRL+PAGE DOWN.

Figure 4.4 Workspace Window

{ewc msdncd, EWGraphic, jug17b 0 /a "uguideWKSPW.BMP"}

Each pane contains a hierarchical (tree) view consisting of various nodes. You can expand the nodes in the hierarchy to display their contents, or collapse the nodes to display the organization. The top-level node (or nodes) in a pane is the *folder*. Each folder can contain a variety of items. Some items are *container* items, such as a classes file, which contains a list of the classes contained

in the project. Container items can also be expanded. A bottom-level node, which you cannot further expand, represents an *editable item*. When you open an editable item, it opens the appropriate editor—text editors for source files or classes, dialog editor for dialog box resources, and so on—to edit the resource.

Tip While using any pane in the Project Workspace window, you can click the right mouse button when the mouse pointer is over the selection to display a pop-up menu of frequently used commands. The available commands depend on the current selection. For example, if the selection is a source file, the pop-up menu shows the Properties command and other commands to manipulate the pane.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Folders

The top-level node (or nodes) in a pane is the *folder*. Each folder can contain a variety of items, including other folders. You can open a folder to display the items that it contains, or you can close a folder to simplify the view in the pane.

The types of panes displayed in the Project Workspace window depend on the type of project. In Visual J++ projects, for instance, Microsoft Developer Studio displays a pane that contains the classes in the project.

{ewl msdncd, EWGraphic, jug18b 0 /a "build.bmp"} To open a folder

- Double-click the folder.
—or—
Click the plus sign (+) to the left of the folder.

{ewl msdncd, EWGraphic, jug18b 1 /a "build.bmp"} To close a folder

- Double-click the folder.
—or—
Click the minus sign (–) to the left of the folder.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Items

Each folder in a pane can contain a variety of items. Some items are *container* items, such as a class file, which contains a list of the classes contained in the project. Container items can be expanded in the same way folders can be expanded. A bottom-level node that you cannot expand further represents an *editable item*. When you open an editable item, it opens the appropriate editor—text editors for source files or classes, dialog box editor for dialog box resources, and so on—to edit the resource. All items have properties, which you can view and edit on an item's property page(s). Each type of item has a distinct set of properties.

{ewl msdncd, EWGraphic, jug19b 0 /a "build.bmp"} To open an editable item

- Double-click the item.
The appropriate editor for the item opens and displays the item.

{ewl msdncd, EWGraphic, jug19b 1 /a "build.bmp"} To view or change an item's properties

- Select the item, and then press ALT+ENTER.
—or—
Select the item, click the right mouse button, and from the pop-up menu, choose Properties.
—or—
Select the item, and from the Edit menu, choose Properties.

The property page for the item appears. If the item has editable properties, you can edit them on the property page, and those edits take immediate effect.

{ewl msdncd, EWGraphic, jug19b 2 /a "build.bmp"} To delete files from a folder

- Select the item, and press the DEL key.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Shortcut Methods for Views

While using any view in the Project Workspace window, you can click the right mouse button when the mouse pointer is over the selection to display a pop-up menu of frequently used commands. The commands available depend on the current selection. For example, if the selection is a source file, the pop-up menu shows the Properties command and other commands to manipulate the pane.

You can use the shortcut methods listed in Table 4.1 to navigate in the various views, to expand and contract nodes, to select items, and so on.

Table 4.1
Shortcut
Methods
for Views
Method **Result**

HOME	Moves to first node in tree
END	Moves to last node in tree
PAGE UP	Moves up one page (number of visible items determines page size)
PAGE DOWN	Moves down one page (number of visible items determines page size)
CTRL+PAGE UP	Activates previous Project Workspace window pane
CTRL+PAGE DOWN	Activates next Project

	Workspa ce window pane
UP ARROW	Moves to previous node in list
DOWN ARROW	Moves to next node in list
CTRL+UP ARROW	Moves focus up one item
CTRL+DOWN ARROW	Moves focus down one item
SHIFT+UP ARROW	Extends selection up one item
SHIFT+DOWN ARROW	Extends selection down one item
LEFT ARROW	Collapse s current node if possible; otherwis e, moves to parent node
RIGHT ARROW	Expands current node if possible; otherwis e, moves to first child node
ENTER	Perform s default action on node (opens/c

	loses folder, opens item in the editor, and so on)
PLUS SIGN	Expands current node if expanda ble
MINUS SIGN	Collapse s current node if collapsib le
ASTERISK	Fully expands current node, including all child nodes
Click plus sign	Expands current node if expanda ble
Click minus sign	Collapse s current node if collapsib le
Double- click	Perform s default action on node (opens/c loses folder, opens item in the editor, and so on)
CTRL+click	Selects or deselect s item (noncont iguous

selection
)
SHIFT+click Selects
block
from
current
selection
to item
(contigu
ous
selection
)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using FileView

The FileView pane shows relationships among the source files and the dependent files used to build all project configurations included in the project workspace. The relationships in FileView are logical relationships, not physical relationships, and do not reflect the organization of files on your hard disk. FileView also shows subprojects within the project workspace, if any exist.

The default project configuration in the workspace is indicated in FileView by bold type. You can select the default configuration by using the Set Default Project Configuration drop-down list from the Project toolbar.

When you expand the top-level folder in FileView, it displays the files included in the project. Figure 4.5 shows an expanded FileView.

Figure 4.5 The FileView Pane

{ewc msdncd, EWGraphic, jug21b 0 /a "uguideFVIEW.BMP"}

FileView uses file icons to convey additional information about the files in the project. Table 4.2 shows the icons and their meanings.

Table 4.2 File Icons in FileView	
Icon	Meaning
{	Developer
e	Studio can use
w	this file in a
c	<u>build</u> , and it is
m	included in the
s	<u>build</u> for this
d	project.
n	
c	
d	
,	
E	
W	
G	
r	
a	
p	
h	
ic	
,	
j	
u	

g
2
1
b
1
/
a
"
u
g
u
i
d
e
B
L
D
I.
B
M
P
"}
{ Developer
e Studio can use
w this file in a
c [build](#), but it is
m not included in
s the [build](#) for this
d project.
n
c
d
,
E
W
G
r
a
p
h
ic
,
j
u
g
2

1
b
2
/
a
"
u
g
u
i
d
e
B
L
D
I
2
.
B
M
P
"}
{
e
w
c
m
s
d
n
c
d
,
E
W
G
r
a
p
h
ic
,
j
u
g
2

Developer
Studio uses this
file as an explicit
dependency in a
project.

1
b
3
/
a
"
u
g
u
i
d
e
D
E
P
I.
B
M
P
"}
{
e
w
c
m
s
d
n
c
d
,
E
W
G
r
a
p
h
ic
,
j
u
g
2
1
b

Developer
Studio cannot
[build](#) this file
using the default
tools. Files in
this category
might include
documentation
or specifications.
You could
specify custom
tools for these
files.

4
/
a
"
u
g
u
i
d
e
N
B
L
D
.
B
M
P
"}
}

{ Developer
e Studio refers to
w this project as a
c subproject of the
m project that
contains it.

s
d
n
c
d
,
E
W
G
r
a
p
hi
c,
ju
g
2
1
b
5
/

a
"
u
g
ui
d
e
S
P
R
O
J.
B
M
P
"}
}

If you have installed a source-code control system that conforms to the Microsoft Common Source Code Control Interface, the icons also represent some source-code control states. A grayed icon indicates that a file is under source-code control. A check mark next to the icon for a file under source-code control indicates that you have the file checked out.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using ClassView

Visual J++ derives the ClassView pane from the contents of the source files included in the project workspace. It shows all the Java classes for which definitions are available, and the members of those classes. The relationships in ClassView are logical relationships, not physical relationships.

Note Visual J++ computes the contents of ClassView as a background process. This may mean that there is some delay from the time you open a project workspace or save a revised file until the view is ready to be displayed. If you are completing other processes that use significant computing resources, the delay may increase.

In ClassView, you can:

- Add methods to the selected class.
- Add variables to the selected class.
- Add a new class to the project.
- Set a breakpoint on a method.

The folder name shown in bold type in ClassView represents the default project configuration. When you expand the top-level folder in ClassView, it displays the classes included in that project. If you expand any class, it displays the members in that class. Figure 4.6 shows an expanded ClassView.

Figure 4.6 ClassView

{ewc msdncd, EWGraphic, jug22b 0 /a "uguideCLSV.BMP"}

ClassView uses icons to convey additional information about the classes and class members in the project. Table 4.3 shows the icons and their meanings.

Table 4.3 Icons in ClassView

Icon	Meaning
{e	Class
w	
c	
m	
s	
d	
n	
c	
d,	
E	
W	
G	
ra	
p	
hi	
c,	

ju
g
2
2
b
1
/a
"
u
g
ui
d
e
C
L
S
I.
B
M
P
"}
{e Default(Package)
w [method](#)
c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
2
/a
"
u

g
ui
d
e
P
K
G
F.
B
M
P
"}
{e

Private [method](#)

w
c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
3
/a
"

u
g
ui
d
e
P
R
V
F.
B

M
P
"}
{e Protected
w [method](#)
c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
4
/a
"
u
g
ui
d
e
P
R
O
T
F.
B
M
P
"}
{e Public [method](#)
w
c
m
s

d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
5
/a
"
u
g
ui
d
e
P
U
B
F.
B
M
P
"}
{e
w
c
m
s
d
n
c
d,
E
W
G
ra
p

Default(Package)
[variable](#)

hi
c,
ju
g
2
2
b
6
/a
"
u
g
ui
d
e
P
K
G
M
.
B
M
P
"}
{e Private [variable](#)
w
c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
7

/a
"
u
g
ui
d
e
P
R
V
M

.
B
M
P
"}
{e

Protected
w [variable](#)

c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
8
/a
"

u
g
ui
d
e
P

R
O
T
M

.
B
M
P
"}
{e

Public [variable](#)

w
c
m
s
d
n
c
d,
E
W
G
ra
p
hi
c,
ju
g
2
2
b
9
/a
"

u
g
ui
d
e
P
U
B
M
.
B
M
P

"}
"

You can group the items in a class either alphabetically by name or alphabetically in access specifier groups—that is, default(package), private, protected, or public.

{ewl msdncd, EWGraphic, jug22b 10 /a "build.bmp"} To group members in a class

- 1 Select one or more class nodes.
- 2 Click the right mouse button to display the pop-up menu.
- 3 Choose Group By Access to toggle the grouping.

If the command has a check mark, the members are already grouped by access specifier; if not, they are grouped alphabetically.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding a Method from ClassView

From ClassView, you can add a method to a selected class. This allows you to readily add methods to an existing class definition.

{ewl msdncd, EWGraphic, jug23b 0 /a "build.bmp"} To add a method

- 1 Select the class to which you want to add a method.
- 2 With the mouse pointer over the selected class, click the right mouse button to display the pop-up menu, and choose Add Method.

The Add Method dialog box appears.

- 3 In the Return Type text box, type the method's return type.
- 4 In the Method Declaration text box, type the method declaration. Type only the method name, followed by a list of the names and types of formal parameters enclosed in parentheses.
- 5 Select an access specifier for the method from the Access drop-down list box.
- 6 Select the modifier or combination of modifiers that you want for your method.

For example, if you want a static method, select the Static check box. If you want your method to be static and synchronized, select the Static and Synchronized check boxes.

Note Visual J++ will allow only valid combinations of the modifiers to be used together. When you select a modifier, those modifiers that are mutually exclusive from the selected modifier will be greyed out. For example, Abstract and Final modifiers may not be used together and this incompatibility is dynamically displayed as the modifiers are selected.

- 7 Choose OK.

The Full Declaration line at the bottom of the Add Method dialog dynamically displays the information you enter. Use this information to verify your method's return type, name, parameters, and modifiers before choosing OK.

This procedure adds a corresponding method body in the .JAVA file for the class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding a Variable from ClassView

From ClassView, you can add a variable to the selected class. This allows you to add variables that are not provided by Java Applet Wizard.

{ewl msdncd, EWGraphic, jug24b 0 /a "build.bmp"} To add a variable

- 1 Select the class to which you want to add a variable.
- 2 With the mouse pointer over the selected class, click the right mouse button to display the pop-up menu, and select Add Variable.

The Add Variable dialog box appears.

- 3 In the Variable type text box, enter the Variable type.
- 4 In the Variable name text box, enter the variable name.
- 5 In the Initial value text box, enter an initializing value for the variable.
- 6 Select an access specifier for the variable from the options in the Access drop-down list box.
- 7 Select the modifier or combination of modifiers that you want for your variable.

For example, if you want a static variable, select the Static check box. If you want your variable to be static and final, select the Static and Final check boxes.

Note Visual J++ will allow only valid combinations of the modifiers to be used together. When you select a modifier, those modifiers that are mutually exclusive from the selected modifier will be greyed out. For example, Final and Volatile modifiers may not be used together and this incompatibility is dynamically displayed as the modifiers are selected.

- 8 Choose OK.

The Full Declaration line at the bottom of the Add Variable dialog dynamically displays the information you enter. Use this information to verify your variable's type, name, initial value, and modifiers before choosing OK.

This procedure adds a definition for the variable to the .JAVA file for the class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding a New Class from ClassView

From ClassView, you can add a new class to the current project. This allows you to add classes that are not provided by Java Applet Wizard.

{ewl msdncd, EWGraphic, jug25b 0 /a "build.bmp"} To add a class

- 1 Select the Java project folder to which you want to add a new class.
—or—
Select a class folder within the expanded view of the project folder.
- 2 With the mouse pointer over the selected class, click the right mouse button to display the pop-up menu, and choose Create New Class.
The Create New Class dialog box appears.
- 3 In the Name text box, type the name of the new class to be added to the project.
- 4 From the Extends drop-down list box, select the name of the class that the new class will be derived from — its superclass.
—or—
Type an asterisk (*) in the Extends text box:
 - The Resolve Class dialog box appears.
 - Choose the superclass for your class from the Select Class list box.
 - Choose OK.
- 5 To add the new class to an existing package, type the name of the package in the Package text box.
—or—
To create a package for the new class, type a name in the Package text box.

Note Packages should always be placed in subdirectories below the project directory. Package subdirectories must have the same name as the package. Microsoft Developer Studio creates this directory structure when you add a package from the development environment. It is important that you maintain the same directory structure if you add packages to your project outside the Developer Studio environment.
- 6 In the Modifiers group, select the appropriate modifier for your class.
For example, if you want a public class, select the Public check box. If you want your class to be public and final, select the Public and Final check boxes.

Note Visual J++ allows only valid combinations of the modifiers to be used together. When you select a modifier, modifiers that are mutually exclusive from the selected modifier will be greyed out. For example, Abstract and Final modifiers may not be used together and this incompatibility is dynamically displayed as the modifiers are selected.
- 7 Choose OK.

This procedure adds skeleton code for the new class to the project and a new class node in the ClassView hierarchy.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Breakpoints in ClassView

From ClassView, you can quickly set breakpoints for use in the integrated debugger. You can set breakpoints on the definition of methods.

{ewl msdncd, EWGraphic, jug26b 0 /a "build.bmp"} To set a breakpoint

- Select the method, click the right mouse button to display the pop-up menu, and choose Set Breakpoint.

The breakpoint is set at the definition, and the breakpoint symbol will appear in the source file at the breakpoint location when debugging begins.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using InfoView

The InfoView pane shows the organization of Books Online. You can display any topic in its hierarchy. InfoView is described fully in the section of Books Online titled [Using InfoViewer](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Projects

A project consists of a project configuration and a set of files, which together determine the final binary output file that you create. Developer Studio creates the final output file from the following elements:

- Settings for the platform for which you are building, such as the locations and names of libraries.
- Settings for the type of .CLASS output file.
- Tools—compiler and so on—required to build for the specified platform, as well as their settings.
- The set of source files.

The information for building each individual project is stored in the makefile for the project workspace, along with the information for all other projects in the workspace.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Inserting and Deleting Projects

You can insert new projects into your project workspace. You could, for instance, create an initial Java Workspace project with an applet, and add a second Java Workspace project with another Java applet.

{ewl msdncd, EWGraphic, jug29b 0 /a "build.bmp"} To insert a new project into an existing project workspace

- 1 From the Insert menu, choose Project.
The Insert Project dialog box appears, with the Java project option selected.
- 2 In the Name text box, type a name for the project.
This name is appended to the existing project workspace directory to form the fully qualified path for the new project directory.
- 3 From the Type list, select Java Workspace or Java Applet Wizard.
- 4 Choose Create.

The new project that you just created becomes the default project in the project workspace. If you chose a Java Workspace project type you need to add files to your project. You can then build your new project by choosing Rebuild All from the Build menu.

{ewl msdncd, EWGraphic, jug29b 1 /a "build.bmp"} To delete a project from a project workspace

- 1 From the Build menu, choose Configurations.
The Configurations dialog box appears.
- 2 In the Projects And Configurations box, expand the project that you want to delete.
Projects are the leftmost entities in the tree. You can click the plus or minus sign to expand or contract them to show or hide their configurations.
- 3 Select each configuration in the project you want to delete in turn, and choose Remove.
Respond Yes to the message box that appears each time. When you have removed the last configuration, the project is removed as well.
- 4 Choose Close.

Note Deleting a project removes it as a subproject from any other project in the project workspace.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding and Removing Files from Projects

When you add a file to a project, you add the file to all project configurations in that project. For instance, if you have a project named MyProject, with Debug and Release configurations. Adding a file to your project adds it to these project configurations.

{ewl msdncd, EWGraphic, jug30b 0 /a "build.bmp"} To add files to a project

- 1 From the Set Default Project Configuration drop-down list on the Project toolbar, select the project to which you want to add files.
If the Project toolbar is not displayed, choose Toolbars from the View menu, and select Project from the list.
- 2 From the Insert menu, choose Files Into Project.
The Insert Files Into Project dialog box appears.
- 3 Select the file type to display.
- 4 If necessary, select the drive and directory to view.
- 5 Select one or more files from the File Name list. You can use the SHIFT or CTRL key in conjunction with the mouse to make multiple selections.
- 6 Choose OK.

This procedure adds the files to the selected project.

Repeat the steps for all types of files that you want to add, or to add files from different subdirectories.

If you create a new source file, or open a source file that is not included in the current default project, you can quickly add it to a project with the pop-up menu.

{ewl msdncd, EWGraphic, jug30b 1 /a "build.bmp"} To add an open source file to a project

- 1 With the mouse pointer in the source file, click the right mouse button.
- 2 From the pop-up menu, choose Add To Project, and select the project name from the cascading menu.

{ewl msdncd, EWGraphic, jug30b 2 /a "build.bmp"} To remove files from a project

- Select the file in FileView, and from the Edit menu, choose Delete.
—or—
Press the DEL key.

You can hold down the CTRL or SHIFT keys and use the mouse to select multiple files in the Project Workspace window.

Tip Press CTRL and click a selection to toggle the selection state for the clicked item. You can use this method to quickly remove a file from a multiple selection.

{ewl msdncd, EWGraphic, jug30b 3 /a "build.bmp"} To move or copy files from one project workspace to another

- 1 In the FileView pane of the Project Workspace window, select the files that you want to move or copy.
You can hold down the CTRL or SHIFT keys to select multiple files in the Project Workspace

window.

- 2 From the Edit menu, choose Cut if you want to move the files, or Copy if you want to copy the files.
- 3 Close the current project workspace.
- 4 Open the destination project workspace.
- 5 Select the project to receive the files.
- 6 From the Edit menu, choose Paste.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating and Deleting Configurations in a Project

A project configuration consists of settings that determine the characteristics of the final output file for a project. When you create a new project configuration for a project, it initially has the settings from an existing project configuration. The new project configuration always uses the same set of files as that existing project configuration. You must also specify a platform for the configuration.

A new configuration is a way to make a variation of a project that you are currently building. It could merely specify a different platform, or it could specify different optimization options, for instance.

{ewl msdncd, EWGraphic, jug31b 0 /a "build.bmp"} To create a project configuration

- 1 From the Build menu, choose Configurations.
The Configurations dialog box appears.
- 2 In the Projects And Configurations box, select the project to which you want to add a configuration.
Projects are the leftmost entities in the tree. You can click the plus or minus sign to expand or contract them to show or hide their configurations.
- 3 Choose Add.
The Add Project Configuration dialog box appears.
- 4 In the Configuration text box, type a new name.
This name, along with the platform type, will be used to identify the new configuration.
- 5 From the Copy Settings From drop-down list, select the configuration from which the new configuration copies its initial settings.
- 6 From the Platform drop-down list, select a platform for the new configuration.
You can select the same platform as the project on which you are basing the new one if you want merely a variation of the existing settings.
- 7 Choose OK.
The Configurations dialog box reappears.
- 8 Choose Close.

A new project configuration is now available from the Set Default Project Configuration drop-down list on the Project toolbar. You can now choose new project configuration settings for this configuration, and those settings will be retained in the project configuration.

Note If you choose settings incompatible with the project type for the project or platform on which you based this configuration, you may not get the result that you expect.

If you add files to the project containing this configuration, those files are also used to build the configuration.

{ewl msdncd, EWGraphic, jug31b 1 /a "build.bmp"} To delete a configuration from a project

- 1 From the Build menu, choose Configurations.
The Configurations dialog box appears.
- 2 In the Projects And Configurations box, expand the project from which you want to delete a configuration.

Projects are the leftmost entities in the tree. You can click the plus or minus sign to expand or contract them to show or hide their configurations.

3 Select the configuration that you want to remove.

4 Choose Remove.

Respond Yes to the message box that appears.

5 Choose Close.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Updating Dependent Files in a Project

After editing one or more source files, you can explicitly update the project to synchronize dependent files with their dependencies.

{ewl msdncd, EWGraphic, jug32b 0 /a "build.bmp"} To update dependent files in a Visual J++ project

- From the Build menu, choose Rebuild All.

Note Do not choose Update All Dependencies. Microsoft Developer Studio provides the Update All Dependencies command to search for and update #include directives in a Visual C++ project. Since Java does not use #include statements, the Update All Dependencies will not give you the results you expect when used on a Visual J++ project.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting the Class Path Directories

A classpath value is registered by the Virtual Machine (VM) as C:\WINDOWS\Java\Classes\Classes.zip;.:. The default root directory is C:\WINDOWS if you are using Microsoft Windows 95, but could be another directory, such as, C:\WINNT, if running on Microsoft Windows NT (version 4.0 or later). When a Java program runs, both Visual J++ and the VM look for the classpath value in HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM\Classpath registry key. The value in this key is the default CLASSPATH environment variable.

If you specify other directories for your classes from the Options dialog on the Tools menu (see Setting Directories), Visual J++ and the VM will also search for classes in those directories.

You can specify an additional directory where project-specific classes are located. This allows you to easily add a directory to be searched in addition to any directories listed in the Directories tab of the Options dialog from the Tools menu and the default CLASSPATH environment variable stored in the registry.

{ewl msdncd, EWGraphic, jug33b 0 /a "build.bmp"} To set an additional Class Path Directory

- 1 From the Build menu, choose Settings.
The Project Settings dialog box appears, as shown in Figure 4.7.
- 2 In the Settings For pane, select the project node for which you want to set an additional class directory.
- 3 Select the General tab.
The General tab is one of three that contain options for the project. This tab specifies which directories the project uses for an additional class path and final output files.
- 4 In the Class path directories text box, type the directory name for the .CLASS files.
- 5 Choose OK.

Figure 4.7 Project Settings Dialog Box

{ewc msdncd, EWGraphic, jug33b 1 /a "uguidePRJSET.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Selecting the Directories for Output Files

You can select the directories in which to put the final output files for each project configuration. By putting these files in different directories, you can maintain copies of the same files built in different ways—for instance, the Debug and the Release versions of your project.

{ewl msdncd, EWGraphic, jug34b 0 /a "build.bmp"} To select output directories

- 1 From the Build menu, choose Settings.

The Project Settings dialog box appears, as shown in Figure 4.7.

- 2 In the Settings For pane, select the project node for which you want to set directories.
- 3 Select the General tab.

The General tab is one of three that contain options for the project. This tab specifies which directories the project uses for an additional class path and final output files.

- 4 Type the directory name for the final output files (.CLASS files, for instance) in the Output directory text box.
- 5 Choose OK.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Specifying Project Configuration Settings

You can set options for a project configuration only when it is selected.

`{ewc msdncd, EWGraphic, jug35b 0 /a "uguide.BMP"}` **To specify project settings**

- 1 From the Build menu, choose Settings.

The Project Settings dialog box appears, as shown in Figure 4.7.

- 2 In the Settings For pane, select the project configuration, such as Java Virtual Machine Debug shown in Figure 4.7.

You can also select multiple project configurations, and specify settings common to all the configurations.

- 3 From the tabs at the top of the dialog box, select the type of settings that you want to specify.

- 4 Specify the settings you want on the selected tab.

When you have completed specifying the settings on a tab, you can select another and specify additional settings. CTRL+TAB displays the next tab, and CTRL+SHIFT+TAB displays the previous tab.

- 5 When you have completed setting options, choose OK.

Note If you specify settings incompatible with the project type that you chose when you created your project, you may not get the result that you expect.

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

Excluding a File from the Build

By default, all files in a project are marked to be built in the project configuration file. A file is built with a project configuration's settings when you build that configuration. However, you can exclude an individual file from being built.

{ewl msdncd, EWGraphic, jug36b 0 /a "build.bmp"} To exclude a file from being built

- 1 From the Build menu, choose Settings.
The Project Settings dialog box appears, as shown in Figure 4.8.
- 2 Select the General tab.
- 3 In the Settings For pane, expand the project, such as NewApp Java Virtual Machine Debug shown in Figure 4.8, and select the file you want to exclude.
Use the SHIFT and CTRL keys with the mouse to make multiple selections.
- 4 Select the Exclude file from build check box on the General tab.
- 5 Choose OK.

You can also specify common settings across multiple projects or project configurations. Within the projects in your project workspace, you can select any combination of files.

Figure 4.8 Exclude File From Build

{ewc msdncd, EWGraphic, jug36b 1 /a "uguideEXCL.BMP"}

{ewl msdncd, EWGraphic, jug36b 2 /a "build.bmp"} To specify file settings in multiple project configurations

- 1 From the Build menu, choose Settings.
The Project Settings dialog box appears, as shown in Figure 4.8.
- 2 In the Settings For pane, expand the project configurations, such as the Debug and Release configurations of NewApp Java Virtual Machine shown in Figure 4.8, and select the file or files.
Click the plus signs or double-click the node names to expand the graph of project files if necessary, and use the SHIFT and CTRL keys with the mouse to make multiple selections.
- 3 Select the Exclude file from build check box on the General tab.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Specifying Custom Build Tools

You can specify custom build tools for use with any project or with any individual files that do not already have a tool associated with them. These tools then process the files at the appropriate point in the build if the output file is out of date with respect to the input file. For instance, you can add an .L file to your project, specify a lexical analyzer to process the file and produce a .Y output file, and then specify a parser generator to process that file to create a Java source-code file for Visual J++. You could also select the output file for a configuration, to copy it to a specific directory for testing, for instance. Microsoft Developer Studio provides a number of macros for use in these commands.

Note By default, a .JAVA file type has tools associated with it in Visual J++. You cannot specify a custom tool for .JAVA files.

You can specify more than one custom tool for a file or project; the tools run in the order that you specify them.

The custom tools run on files only in builds of the configurations in which you selected the files. That is, if your file set includes an .L file, and you select it only in one configuration, the tools that you specify run only in that configuration.

{ewl msdncd, EWGraphic, jug37b 0 /a "build.bmp"} To specify custom build tools

- 1 From the Build menu, choose Settings.
The Project Settings dialog box appears.
- 2 In the Settings For pane, select the source files or output files from project configurations for which you want to specify a custom tool or tools.
Selecting the top-level node specifies the output file for a configuration.
- 3 Select the Custom Build tab.
If you have made multiple selections, the Input File text specifies multiple selections.
- 4 In the Description text box, type a description.
This description appears on the Build tab of the Output window when the command runs.
- 5 In the Build Command(s) list, select the first line, and type the command that you want to run on the input file.
If you type more than one command in the grid, the build process runs them in order, from top to bottom.

Note The command must include all required options, including the input file name or names and output file name or names. You may want to use a directory macro to specify the location for the output file.
- 6 In the Output File(s) list, select the first line, and type the name of an output file that is created by the build commands specified in the Build commands grid.
If the commands create more than one output file, type additional names in the subsequent lines of the grid.
- 7 Choose OK.

For example, assume that you want to include in your project a file named MYLEXINF.L. You first want a lexical analyzer to process MYLEXINF.L to produce a .Y file with the same base name (MYLEXINF.Y). You then want a parser generator to process MYLEXINF.Y to produce a .C file.

First, you add MYLEXINF.L and MYLEXINF.C to your project using the Files Into Project command on the Insert menu. (If you have not already created a version of MYLEXINF.C, Microsoft Developer Studio recognizes that and asks if you want to add a reference to the file anyway.) You then choose Settings command from the Build menu, and select MYLEXINF.L in the appropriate configuration. Next, select the Custom Build tab, and type commands similar to the following in the Build Command(s) list:

```
lexer $(InputPath) $(IntDir)\$(InputName).y  
parser $(IntDir)\$(InputName).y $(ProjDir)\$(InputName).c
```

This puts the intermediate output, MYLEXINF.Y, in the directory used for intermediate files, and generates MYLEXINF.C in the project directory.

In the Output File(s) list, type \$(ProjDir)\\$(InputName).C. When you build this project, the build system checks the date of MYLEXINF.C. If its date is earlier than MYLEXINF.L, the build system runs these custom commands to rebuild MYLEXINF.C.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Macros for Custom Build Commands

You can use the File and Directory drop-down lists to insert any of the following directory and filename macros in either grid at the current insertion point location. The File and Directory drop-down lists are on the Custom Build tab of the Project Settings dialog box, accessed with the Settings command from the Build menu.

Label	Macro	Description
Intermediate	\$ (InterDir)	Path to the directory specified for intermediate files, relative to the project directory
Output	\$ (OutputDir)	Path to the directory specified for output files, relative to the project directory
TARGET	\$ (TargetDir)	Fully qualified path to the directory specified to output files
Input	\$ (InputDir)	Fully qualified path to the project directory
Project	\$ (ProjectDir)	Fully qualified path to the project directory
Works	\$ (Works)	Fully qualified

pac e	pDir)	path to the project directory
Mic ros oft De vel ope r	\$ (MSDevDir)	Fully qualified path to the installation directory for Microsoft Developer Studio
Re mot e Tar get	\$ (RemoteDir)	Fully qualified path to the remote output file
Tar get Pat h	\$ (TargetPath)	Fully qualified name for the project output file
Tar get Na me	\$ (TargetName)	Base name for the output file
Inp ut Pat h	\$ (InputPath)	Fully qualified name for the input file
Inp ut Na me	\$ (InputName)	Base name for the input file
Wo rks pac e Na me	\$ (WksProjectName)	Name of the project workspace workspace
Re mot e Tar get	\$ (RemoteTargetPath)	Fully qualified name for the remote

Pat output
h file

If you have made multiple selections, during a build the input macros are set in turn to each file that you have selected for each configuration that you have selected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Building a Project Configuration

From Microsoft Developer Studio, you can build or rebuild the program that a project configuration defines. When you build a project configuration, Developer Studio processes only the files in the project that have changed since the last build. When you rebuild a project configuration, Developer Studio processes all the files in the project. You can either choose to build a single project configuration, or the default project configuration, or you can choose multiple configurations to build in one operation.

When you create a project, Developer Studio sets default options for both Debug and Release configurations. The Debug configuration contains full symbolic debugging information that can be used by the integrated debugger in Developer Studio or by other debuggers that use the Microsoft debug format. Developer Studio also turns off all optimizations in the Debug configuration because they generally make debugging more difficult. The Release configuration does not contain any symbolic debugging information, and it uses any optimizations that you have set after creating the projects. Depending on your installation and the choices you made when you created your project, you may have other default project configurations with other options, or you may have specifically created other project configurations with other options. Each project configuration also specifies the directories in which the final files are created.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting the Default Project Configuration

When you set the default project configuration, subsequent build commands act on the default configuration and build its output. If the project associated with the default configuration contains subprojects, the same configuration in the subprojects gets built if the output file for the configuration in the subproject is out of date.

{ewl msdncd, EWGraphic, jug40b 0 /a "build.bmp"} To set the default project configuration

- 1 From the Build menu, choose Set Default Configuration.
The Default Project Configuration dialog box appears.
- 2 In the Project Configurations list, select the default project configuration.
- 3 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Building the Default Project Configuration

You can choose the project configuration that you want to build by default. This is the project configuration that you build when you choose Build *project* from the Build menu. *Project* represents the program defined by the project configuration. If this project configuration contains any explicit project dependencies, and those project configurations are out of date, they are built first.

{ewl msdncd, EWGraphic, jug41b 0 /a "build.bmp"} To select a project configuration

- 1 From the Build menu, choose Set Default Configuration.
The Default Project Configuration dialog box appears.
- 2 In the Project Configurations list, select the default project configuration.
- 3 Choose OK.

{ewl msdncd, EWGraphic, jug41b 1 /a "build.bmp"} To build the default project configuration

- From the Build menu, choose Build *project*, where *project* represents the program or library defined by the project configuration.

If you want to ensure that all files associated with a project configuration get built, whether or not they are out of date, you can choose the Rebuild All command.

{ewl msdncd, EWGraphic, jug41b 2 /a "build.bmp"} To rebuild the default project configuration

- From the Build menu, choose Rebuild All.

Information about the build is displayed in the Output window. The Output window displays information from the build tools and lists any errors or warnings that occur during the build. If no errors are reported, the build completed successfully. If errors are reported, you need to debug them. For information on debugging build errors, see Debugging Compiler Errors.

{ewl msdncd, EWGraphic, jug41b 3 /a "build.bmp"} To stop a build

- From the Build menu, choose Stop Build.

Developer Studio stops the currently executing tool if possible; otherwise, it stops the build as soon as the currently executing tool finishes.

Since builds occur in the background, you can continue to use Developer Studio during a build. However, some menu commands and toolbar buttons are disabled during a build. You can use the tabs at the bottom of the Output window to view the previous output from another tool while you are running the current build, and then choose the Build tab to return to the current build output.

An audible message notifies you when the build is complete. Unless you have a sound card installed, all audible events issue a beep. If you have a sound card installed, you can use the Sound application in the Windows Control Panel to assign the three standard system events listed below to different sounds.

**Syst Indicates
em
Eve**

nt

Aste Build has
risk (completed
*) without errors
or warnings.

Que Build has
stion completed
(?) with warnings.

Excl Build has
ama completed
tion with errors.
(!)

In some cases, you may need to stop building your project before the process finishes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiling Files

You can select and compile files in any project in your project workspace.

{ewl msdncd, EWGraphic, jug42b 0 /a "build.bmp"} To compile selected files

- 1 Select the files in the FileView pane of the Project Workspace window.
- 2 With the mouse pointer over the selection, click the right mouse button to display the pop-up menu, and choose Build.

If you have specified a custom tool (or tools) for a file, when you select Compile, Developer Studio runs that tool with the file as input, and produces the output specified.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Building Multiple Project Configurations

Any project workspace can have more than one project configuration. Instead of selecting each project configuration in turn and building it as the default configuration using the Build *project* command on the Build menu, you can select multiple project configurations and build them all.

Note You can also build multiple project configurations by using subprojects, and building the appropriate containing project. See Using Project Workspaces: Two Basic Scenarios for more information.

{ewl msdncd, EWGraphic, jug43b 0 /a "build.bmp"} To build multiple project configurations

- 1 From the Build menu, choose Batch Build.
The Batch Build dialog box appears. By default, all project configurations in the project workspace are selected.
- 2 If you don't want to build certain project configurations, clear the check boxes in the Project Configurations list.
- 3 Choose Build to build only those intermediate files of each project configuration that are out of date, or the Rebuild All button to build all intermediate files for each project configuration.
The results for each project configuration are separated in the Output window by a line containing the name of the project configuration being built.

{ewl msdncd, EWGraphic, jug43b 1 /a "build.bmp"} To stop building multiple projects

- From the Build menu, choose Stop Build.

Developer Studio stops the currently executing tool if possible; otherwise, it stops the build as soon as the currently executing tool finishes. The build of the project configuration currently in progress ends. A message box appears, asking if you wish to continue building the remaining project configurations. If you choose Yes, then the batch build continues from the next configuration in the list. If you choose No, then the entire batch build is stopped.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Opening an Existing Makefile

When you open an existing makefile (MAKEFILE or *filename*.MAK) that Developer Studio does not recognize as a makefile that it created, it displays a message box asking if you want to create a project to wrap the makefile. If you choose No, Developer Studio does not open the file. If you choose Yes, Developer Studio creates a project workspace and its associated file for the external makefile. The external makefile becomes a part of the project workspace.

{ewl msdncd, EWGraphic, jug44b 0 /a "build.bmp"} To open an existing makefile with the extension .MAK

- 1 From the File menu, choose Open Workspace.

The Open Project Workspace dialog box appears.

- 2 Select All Files from the drop-down list to display all files.
- 3 From the Open As drop-down list, select Makefile.
- 4 Select the drive and directory containing the makefile that you want to open.
- 5 Select the file from the list and choose OK.

—or—

Double-click the filename in the list.

If you have a project workspace currently open, Developer Studio saves the workspace and asks if you want to close document windows associated with that workspace.

Developer Studio displays a message box asking if you want to convert the external makefile into a project workspace with an external project containing the external makefile.

- 6 Choose Yes to convert the makefile.

If you choose No, Developer Studio cancels the conversion process.

- 7 Choose OK.

Developer Studio displays the Save As dialog box, with a default name for the Developer Studio project workspace file.

- 8 Either enter a new name, or accept the default name, and select OK to create the project workspace file. If necessary, choose a drive or directory for the file.

Note You cannot use the name of the existing makefile for the Developer Studio project workspace file. If you used that name, the project workspace file would overwrite the existing makefile, and would then have no file to run.

If your makefile has a different extension, or has the name MAKEFILE, you can use the Open command from the File menu to open it as a makefile.

{ewl msdncd, EWGraphic, jug44b 1 /a "build.bmp"} To open an existing makefile without the extension .MAK

- 1 From the File menu, choose Open.

The Open dialog box appears.

- 2 Select All Files from the drop-down list to display all files.
- 3 Select Makefile from the Open As drop-down list.
- 4 Select the drive and directory containing the makefile that you want to open.
- 5 Select the .MAK file from the File Name list and choose OK.

–or–

Double-click the filename in the list.

If you have a project workspace currently open, Developer Studio saves the workspace and asks if you want to close document windows associated with that workspace.

Developer Studio displays a message box asking if you want to convert the external makefile into a project workspace with an external project containing the external makefile.

- 6 Choose Yes to convert the makefile.

If you choose No, Developer Studio cancels the conversion process.

- 7 Choose OK.

The Save As dialog box appears, with a default name for the Developer Studio project workspace file.

- 8 Either enter a new name, or accept the default name, and select OK to create the project workspace file. If necessary, you can choose a drive or directory for the file.

Note You cannot use the name of the existing makefile for the Developer Studio project workspace file. If you used that name, the project workspace file would overwrite the existing makefile, and would then have no file to run.

Developer Studio opens a Project Workspace window for the project and shows FileView with only the external project-level nodes and the external makefile as a source file in each. You can use the menu commands to add or delete files from these projects. You can also use the Settings command on the Build menu to change the settings for external projects.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Building a Single File Without a Project Workspace

You can create a single source file and then build a stand-alone Java application directly from that source file. This method is generally useful only for relatively simple applications.

{ewl msdncd, EWGraphic, jug45b 0 /a "build.bmp"} To build a stand-alone Java application from a single source file

- 1 Close any open project workspace.
- 2 Create or open a source file in a text editor window.
- 3 From the Build menu, choose Build.

Developer Studio displays a message box asking if you would like to create a project workspace.

- 4 Choose Yes.

The Save As dialog box appears if you have not yet given the source file a name.

- 5 If necessary, give the source file a new name, with an extension for a file that Developer Studio can build, such as .JAVA for Visual J++.
- 6 Choose OK.

Developer Studio creates a default project workspace using the base name of the source file as the base name for the project. It uses default settings for the project configuration and builds the application.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Running an Applet or Application

When you have completed building a project configuration, you can start a Java applet or stand-alone Java application from Microsoft Developer Studio. You can also run these programs in the integrated debugger.

{ewl msdncd, EWGraphic, jug46b 0 /a "build.bmp"} To run a program

- From the Build menu, choose Execute *project*, where *project* represents the program defined by the project configuration.

{ewl msdncd, EWGraphic, jug46b 1 /a "build.bmp"} To run a program in the integrated debugger

- From the Build menu, choose Debug, and from the cascading menu, choose Go, Step Into, or if you have a source file open and it has the fOCUS, Run To Cursor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Text Editor

Microsoft Developer Studio provides an integrated text editor to manage, edit, and print source files. Most of the procedures for using the editor should seem familiar if you have used other Windows-based text editors. With the text editor, you can:

- Use the File menu to:
 - Create source files,
 - Open single files,
 - Open multiple files
 - Save and print source files.
- Move around in a source file
 - With the Go To dialog box
 - By matching group delimiters
 - With bookmarks to mark frequently accessed lines in your source file
 - Using a wide range of navigation commands.
- Perform advanced Find and replace operations in
 - A single file
 - Multiple files
- Use regular expressions with:
 - Developer Studio
 - BRIEF emulation
 - Epsilon emulation
- Manipulate text selections with
 - Cut, copy, paste, and delete text with the Edit menu
 - Undoing and redoing editing actions
 - Using drag-and-drop
 - Specify text selection for lines, multiple lines, and columns.
- Record and play back keystrokes.
- Modify your workspace by
 - Customizing the text editor with save preferences, the selection margin, and tabs and indents
 - Using virtual spaces for advanced cursor positioning
 - Setting the font style, size, and color
- Set syntax coloring for:
 - Source files
 - HTML files
- Control the source window by switching between windows, opening new windows, splitting window views, and using full-screen mode.

Tip While using the text editor you can click the right mouse button to display a pop-up menu (also called a context menu) of frequently used commands. The commands available from the context menu depend on where the cursor is pointing and whether you are in edit or debug mode. For

example, if you click while pointing to the name of a file, the pop-up menu shows a command to open that file, as well as other commands including Open With, Copy, Rename, and Properties.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File Management

The text editor File menu has several commands for standard file management. With these commands you can perform the following actions:

- Create files
- Open files
- Open multiple files
- Save files
- Print files
- Use the File menu to create source files, open single files, open multiple files, and save and print source files.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating Files

The New command creates a new source file. Creating a source file does gives focus to the new file, but does not affect the state of other open source files in any other way.

{ewl msdncd, EWGraphic, jug2c 0 /a "build.bmp"} To create a new source file

- 1 From the File menu, choose New.

The New dialog box appears.

- 2 Select Text File, and then choose OK.

- 3 From the File menu, choose Save.

The Save As dialog box appears.

- 4 Select a path where you want to store the source file.

- 5 In the Filename box, type a filename.

The default extension given to a file is the last extension used when you saved a file. You can type another extension or select one from the Save As Type drop-down list box.

Note If you have not previously saved a file, the default extension will be .TXT. Change the extension to .JAVA if you want to add it to a Visual J++ project.

- 6 Choose Save.

New files are labeled Text n until they are saved. The n is a sequential number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Opening Files

When you open a source file, its name is added to the Window menu. You cannot use the Open command on the File menu to open another copy of an open source file.

{ewl msdn cd, EWGraphic, jug3c 0 /a "build.bmp"} To open a file

- 1 From the File menu, choose Open.

The Open dialog box appears.

- 2 Select the drive and directory where the file is stored.
- 3 If you want read only, select the Open As Read Only check box.

Note You can edit a file even if the Open As Read Only check box is selected. When you save the file, the Save As dialog box appears, allowing you to save the file using a different name.

- 4 Specify the types of files to display in the Files Of Type box.

Files with the chosen extension are displayed in the list box. For example, Project Workspaces displays all files with the .MDP extension. The Files Of Type drop-down list box initially lists commonly used file extensions. The common files default shows the .java, .html extensions.

—or—

Specify wildcard patterns in the Filename box to display file types. You can use any combination of wildcard patterns, delimited by semicolons. For example, if you type *.java;*.html, all files with these extensions are displayed. The wildcard patterns you specify are retained until you close the dialog box.

- 5 Select a filename, then choose the Open button.

—or—

Double-click the filename.

You can also open a file by double-clicking the file icon in the Project Workspace, or by dragging the icon of a non-project file into the application window.

Tip The names of the four most recently opened files are displayed at the end of the File menu. To open one of these files, choose its name from the menu.

Note The number of files on the list of most recently opened files is controlled by the FileCount item in the Registry.

The text editor commands that can open or activate a new file are described in the following table. To get keyboard shortcut information for the following commands choose Keyboard from the Help menu.

Comm and	Descript ion
Bookm ark	Edits or navigate s bookmar ks.
GoTo	Moves to a

	specified location.
GoToErrorTag	Moves to the line containing the current error or tag.
GoToNextErrorTag	Moves to the line containing the next error or tag.
GoToPreviousErrorTag	Moves to the line containing the previous error or tag.
WindowList	Manages the currently open windows.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Opening Multiple Files

You can open multiple files from the Open dialog box by using the mouse to select a file or group of files. Before you can select files, they must be visible in the Directories window.

{ewl msdncd, EWGraphic, jug4c 0 /a "build.bmp"} To open two or more files in sequence

- 1 From the File menu, choose Open.

The Open dialog box appears.

- 2 Select the drive and directory where the files are stored.

The default is the current drive and directory.

- 3 Specify the types of files to display in the Files Of Type drop-down listbox.

Files with the chosen extension are displayed in the list box. For example, Project Workspaces displays all files with the .MDP extension. The Files Of Type drop-down list box initially lists commonly used file extensions. The default shows the .java, .html extensions.

—or—

Specify wildcard patterns in the Filename box to display file types. You can use any combination of wildcard patterns, delimited by semicolons. For example, if you type *.java;*.html, all files with these extensions are displayed. The wildcard patterns you specify are retained until you close the dialog box.

- 4 Click the first file or directory you want to select.
- 5 Hold down the SHIFT key while you click the last file or directory in the group, and then choose the Open button.

{ewl msdncd, EWGraphic, jug4c 1 /a "build.bmp"} To open two or more files out of sequence

- 1 From the File menu, choose Open.

The Open dialog box appears.

- 2 Select the drive and directory where the files are stored.

The default is the current drive and directory.

- 3 Specify the types of files to display in the Files of Type box.

Files with the chosen extension are displayed in the list box. For example, Project Workspaces displays all files with the .MDP extension. The Files of Type drop-down list box initially lists commonly used file extensions. The default shows the .java, .html extensions.

—or—

Specify wildcard patterns in the Filename box to display file types. You can use any combination of wildcard patterns, delimited by semicolons. For example, if you type *.java;*.html, all files with these extensions are displayed. The wildcard patterns you specify are retained until you close the dialog box.

- 4 Hold down the CTRL key while you click each file or directory that you want. Once your selection is complete, choose the Open button.

To cancel a selection, hold down CTRL while you click the selected file or directory.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Saving Files

As you make changes to a source file, an asterisk (*) appears after the filename in the title bar to indicate that the file has changed since it was last saved. Each source window associated with a source file can retain its own sizing and other window attributes.

{ewl msdncd, EWGraphic, jug5c 0 /a "build.bmp"} To save a file

- 1 Switch to the source window.
- 2 From the File menu, choose Save.
If you have already saved the file, the Save command saves changes without displaying the Save As dialog box.
If your file has not been previously saved, the Save As dialog box appears.
- 3 In the Filename box, type a new filename.
- 4 Select the drive and directory where you want to save the file.
- 5 Choose the Save button.

{ewl msdncd, EWGraphic, jug5c 1 /a "build.bmp"} To save all open files

- From the File menu, choose Save All.

{ewl msdncd, EWGraphic, jug5c 2 /a "build.bmp"} To save selected open files

- 1 From the Window menu, choose Windows.
The Windows dialog box appears.
- 2 Select one or more files from the file list.
- 3 Choose the Save button.
- 4 Choose the Cancel button.

You can also save another copy of an existing file. This procedure is useful for maintaining revised copies of a file while keeping the original unchanged.

{ewl msdncd, EWGraphic, jug5c 3 /a "build.bmp"} To save a new file or another copy of an existing file

- 1 Make the file active by clicking the source window.
- 2 From the File menu, choose Save As.
The Save As dialog box appears.
- 3 In the Filename box, type the filename.
- 4 Select the drive and the directory where you want to save the file.
- 5 Choose the Save button .

{ewl msdncd, EWGraphic, jug5c 4 /a "build.bmp"} To set Save options

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Editor tab, and then select the desired save option.
 - To save open files before running any tool, select the Save Before Running Tools check box.
 - To always prompt before saving a file, select the Prompt Before Saving Files check box.
 - To automatically reload externally modified files that have been loaded (but not yet changed)

by the editor, select the Automatic Reload check box.

3 Choose OK.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Printing Files

With the text editor, you can print selected text or a complete file. Text is printed in the default font for the printer if the default editor font is used. Otherwise, the text prints with the selected editor font, if that font is available on the printer.

You can customize your print jobs by adding headers and footers and by adjusting margins.

{ewl msdncd, EWGraphic, jug6c 0 /a "build.bmp"} To print selected text in a source file

- 1 Select the text you want to print.
- 2 From the File menu, choose Print.

The Print dialog box appears. Under Print Range, the Selection option is automatically selected for you.

- 3 Choose OK.

{ewl msdncd, EWGraphic, jug6c 1 /a "build.bmp"} To print a complete source file

- 1 Move the focus to the source file you want to print.
- 2 From the File menu, choose Print.
- 3 The Print dialog box appears. Under Print Range, select the All option is automatically selected for you.
- 4 Choose OK.

{ewl msdncd, EWGraphic, jug6c 2 /a "build.bmp"} To customize a print job

- 1 From the File menu, choose Page Setup.

The Page Setup dialog box appears.

- 2 In the Header and Footer boxes, type the header or footer text, codes, or both. You can use the drop-list to insert codes into the text box. Only one of the alignment options (left, centered, or right) is available at a time for either header or footer.
- 3 Under Margins, type the left, right, top, and bottom measurements.
- 4 Choose OK.

	To print	Use
	File	&f
	me	
	Current	&p
	page	
	number	
	Current	&t
	system	
	time	
	Current	&d
	system	
	date	
	Left	&l
	aligned	
	Center	&c

ed
Right &r
aligned

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Moving Around in Source Files

The text editor provides a variety of methods to move around in a source file. In addition to using the regular mouse movement and page controls, you can:

- Use virtual space for advanced cursor control.
- Identify sections of source code by matching group delimiters.
- Use the Go To dialog box to navigate your source files.
- Set bookmarks to mark frequently accessed lines in your source files.
- Choose from a wide-range of source file navigation commands.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Virtual Space

The Microsoft Developer Studio editor supports virtual space. This means you can move the cursor by one character position. This feature has been implemented in many ways. The most common difference among text editors is whether or not you can move the cursor into a location that does not contain text. For example, if your cursor is on column 20, and there is no text on the line below the current line, moving the cursor down can do one of two things: Either the cursor moves to column 1—because there is no text on the line below—or the cursor remains on column 20. This latter behavior is called virtual space.

When you select the Virtual Spaces option, spaces are inserted between the end of the line and the insertion point before new characters are added to the line. When you clear the Virtual Spaces option, the text editor moves the insertion point to the end of the line.

{ewl msdncd, EWGraphic, jug8c 0 /a "build.bmp"} To enable virtual spaces

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Compatibility tab.
- 3 In the Recommended Options For list box, select the editor emulation in which you want to have virtual spaces.
- 4 Select the Enable Virtual Space check box.
- 5 Choose OK.

Many word processors support the idea of moving the cursor one sentence at a time. Developer Studio's text editor supports this as well (SentenceUp and SentenceDown), but most source code doesn't have the spacing and punctuation marks needed for sentence navigation. Instead, you can use LineUp and LineDown to navigate single lines of source code.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Matching Group Delimiters

Source code is often grouped using delimiters such as `()`, `{}`, and `[]`. These groupings are called levels. You can navigate these levels using the `LevelUp` and `LevelDown` commands. The editor understands nested levels, and matches the correct delimiter even if the level spans several pages and itself contains many levels.

The `LevelUp` command searches backwards for one of the right-side delimiters, and then positions the cursor before the matching left-side delimiter. The `LevelDown` command searches forward for a left-side delimiter, and then positions the cursor after the matching right-side delimiter.

{ewl msdncd, EWGraphic, jug9c 0 /a "build.bmp"} To search forward for a matching level

- Press the `LevelDown` key combination.
The command begins searching for one of the left-side delimiters, which are `(`, `{`, and `[`. When the left-side delimiter is found, the cursor is positioned at the matching right-side delimiter. If a matching delimiter cannot be found, the editor beeps.

The editor also provides the command `GoToMatchBrace`. When the cursor is initially positioned next to a delimiter, the `GoToMatchBrace` command moves the cursor to the matching delimiter in a block. Since this command works independently of whether the character is a right-side or left-side delimiter, you can quickly jump between the start and end of a level.

{ewl msdncd, EWGraphic, jug9c 1 /a "build.bmp"} To move to a matching brace

- 1 Place the insertion point immediately before or after a brace.
- 2 Press the `GoToMatchBrace` key combination.

The insertion point moves forward or backward to the matching brace. Choosing the command again returns the insertion point to its starting place. If a matching brace cannot be found, the editor beeps. This method also works for parentheses, angle brackets, and square brackets.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Go To

The Go To dialog box is organized into three areas: a list of Go To What items, additional selection criteria, and navigation buttons. Depending on the Go To What selection, the additional selection criteria format changes to either an edit control or a list box. You can display Help text in all cases. Items in the following table lists the Go To What types supported by Visual J++ and related additional selection criteria.

Go to what at	Additional selection criteria	Comments
Error/Tag	Enter error/tag	Select one of the listed error/tags.
InfoViewer Annotations	Enter InfoViewer annotated topic	Type the annotated topic.
InfoViewer Bookmarks	Enter InfoViewer bookmark name	Type the bookmark name.
Line	Enter line number	Type the line number

Note Although more Go To What types appear in the Go To dialog, only those listed in the previous table are supported by Visual J++ version 1.0. Because the Go To dialog belongs to Microsoft Developer Studio. Developer Studio, it is used by several package partners, such as, Visual J++ and Visual C++. Each of these products generates specific information that determines what Go To types are supported in the current project. If a Go To What type has not been defined or is not supported by the package partner, the additional selection criteria box is grayed. For example, if you have not defined any bookmarks, the Enter Bookmark Name text box is grayed; since Visual J++ version 1.0 does not generate a .BSR file, there is no support for the Go To What Definition type.

{ewl msdncd, EWGraphic, jug10c 0 /a "build.bmp"} To use the Go To dialog box

- 1 From the Edit menu, choose Go To.
The Go To dialog box appears.
- 2 In the Go To What list box, select the type.
- 3 Enter the additional selection criteria.
- 4 Choose one of the navigation buttons: Go To, Previous, or Next.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Bookmarks

You can set bookmarks to mark frequently accessed lines in your source file. Once a bookmark is set, you can use menu or keyboard commands to move to it. You can remove a bookmark when you no longer need it.

You can use both named and unnamed bookmarks. Named bookmarks are saved between editing sessions. Once you create a named bookmark, you can jump to that location whether or not the file is open. Named bookmarks store both the line number and the column number of the location of the cursor when the bookmark was created. This location is adjusted whenever you edit the file. Even if you delete the characters around the bookmark, the bookmark remains in the correct location.

Unnamed bookmarks are temporary. They are removed when the file containing them is closed or reloaded. Unnamed bookmarks store only the current line, not the column offset of the cursor. When a line containing an unnamed bookmark is deleted, the bookmark is also removed. You can jump to an unnamed bookmark by activating the file and using either the BookmarkNext or BookmarkPrev command. The advantage of unnamed bookmarks is that they are very easy to set (just use BookmarkToggle), and they provide you with visible feedback in the selection margin of your document.

{ewl msdncd, EWGraphic, jug11c 0 /a "build.bmp"} To set a named bookmark

- 1 Move the insertion point to the line and column where you want to set a named bookmark.
- 2 From the Edit menu, choose Bookmark.
The Bookmark dialog box appears.
- 3 In the Name drop-down list box, type the name of the bookmark.
- 4 Choose the Add button to add the named bookmark to the list of bookmarks.
- 5 Choose the Close button.

{ewl msdncd, EWGraphic, jug11c 1 /a "build.bmp"} To remove multiple named bookmarks

- 1 From the Edit menu, choose Bookmark.
The Bookmark dialog box appears.
- 2 In the Name drop-down list box, select the names of the bookmarks to be removed.
- 3 Choose the Delete button to remove the selected bookmarks.
- 4 Choose the Close button.

{ewl msdncd, EWGraphic, jug11c 2 /a "build.bmp"} To go to a named bookmark

- 1 From the Edit menu, choose Bookmark.
The Bookmark dialog box appears.
- 2 In the Name drop-down list box, select the name of the bookmark to go to.
- 3 Choose the Go To button.

{ewl msdncd, EWGraphic, jug11c 3 /a "build.bmp"} To remove a named bookmark

- 1 From the Edit menu, choose Bookmark.
The Bookmark dialog box appears.
- 2 In the Name drop-down list box, select the name of the bookmark to be removed.
- 3 Choose the Delete button to remove the selected bookmark.

4 Choose the Close button.

{ewl msdn cd, EWGraphic, jug11c 4 /a "build.bmp"} To set an unnamed bookmark

- 1 Move the insertion point to the line where you want to set a bookmark.
- 2 Press the BookmarkToggle key combination.

The line is selected, or marked in the margin if you have set the selection margin.

{ewl msdn cd, EWGraphic, jug11c 5 /a "build.bmp"} To move to the next bookmark after the insertion point

- Press the BookmarkNext key combination.

{ewl msdn cd, EWGraphic, jug11c 6 /a "build.bmp"} To move to the previous bookmark before the insertion point

- Press the BookmarkPrev key combination.

{ewl msdn cd, EWGraphic, jug11c 7 /a "build.bmp"} To remove an unnamed bookmark

- 1 Move the insertion point to anywhere on the line containing the unnamed bookmark.
- 2 Press the BookmarkToggle key combination.

The text editor commands that are associated with bookmarks are described in the following table.

Comm and	Descript ion
Bookm ark	Edits or navigate s named bookmar ks.
Bookm arkClea rAll	Clears all unnamed bookmar ks in the window.
Bookm arkNext	Moves to the line containin g the next named or unnamed bookmar k.
Bookm arkPrev	Moves to the line containin g the previous named or unnamed

bookmark.
Bookmarks Toggles
an unnamed
bookmark for the
current line.

Note These bookmark commands can open files or activate different files in the open Windows list. These commands are an easy way to move between text in a number of different source files. For more information on opening files, see [Opening Files](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

The Navigating Commands

The text editor commands for moving around in a source file are described in the following table.

Command	Description
CharLeft	Moves the cursor one character to the left.
CharRight	Moves the cursor one character to the right.
DocumentEnd	Moves the cursor to the end of the file.
DocumentStart	Moves the cursor to the beginning of the file.
GoToIndentation	Moves the cursor to the end of the indentation.
GoToMatchBrace	Finds the matching brace.
Home	Moves the cursor alternately between the beginning of the current

line and
the
beginnin
g of the
text on
that line.

IndentT
oPrev Moves
the
cursor to
the
position
of the
next text
that is on
the
previous
line.

LevelD
own Searches
forward
for the
end of
the next
brackete
d level.

LevelU
p Searches
back for
the
beginnin
g of the
previous
brackete
d level.

LineDo
wn Moves
the
cursor
one line
downwar
d.

LineEn
d Moves
the
cursor to
the end
of the
text on
the
current
line.

LineSta
rt Moves
the
cursor to
the
beginnin
g of the
current

line.

LineUp Moves
the
cursor
one line
upward.

PageD Moves
own the
cursor
one page
downwar
d.

PageU Moves
p the
cursor
one page
upward.

ParaDo Moves
wn the
cursor
forward
to the
beginnin
g of the
next
paragrap
h.

ParaUp Moves
the
cursor
backwar
d to the
beginnin
g of the
previous
paragrap
h.

Senten Moves
ceLeft the
cursor
back to
the
previous
beginnin
g of a
sentence

Senten Moves
ceRight the
cursor
forward
to the
next end
of a

	sentence .
Windo wEnd	Moves the cursor to the bottom of the text window.
Windo wStart	Moves the cursor the the top of the text window.
WordL eft	Moves the cursor backwar d one word.
WordRi ght	Moves the cursor forward one word.

Note The command Home is distinct from the LineStart command. LineStart always moves the cursor to the first column in the line, while Home moves the cursor to different locations depending on the cursor's current location. The Home command moves the cursor to the first non-blank character in the line. However, if the cursor is already located on the first non-blank character, Home moves to the first column of the line.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Finding and Replacing Text

The text editor supports two common searching methods: full string searching and incremental searching. With full string searching, the entire search string is specified before the search begins. With incremental searching, the search is performed as the string is typed.

With the advanced find and replace capabilities of the text editor, you can search for text in a single source file or in multiple files. You can search for literal text strings or use regular expressions to find words or characters. You can even use tagged regular expressions for searching and replacing.

With the find and replace commands of the text editor, you can:

- Find text in a single file.
- Find text in multiple files.
- Replace text.
- Use regular expressions with Developer Studio, BRIEF emulation, and Epsilon emulation.

If you use any of the incremental search commands (IncrementalSearch, IncrementalSearchBack, IncrementalSearchRE, IncrementalSearchREBack), you can modify the search by toggling the word mode (CTRL+W), regular expression mode (CTRL+T), and case sensitive mode (CTRL+C). These keystrokes are not bindable and only affect the incremental search command.

Since IncrementalSearch finds a match while you are typing, you rarely need to type the complete search string. However, if you are looking for a string that occurs within other strings in your file, just repeat the IncrementalSearch command after you have typed enough to specify the particular string you wish to find. Incremental search stops when the ESC key is pressed.

The text editor commands for searching in a source file are described in the following table.

Comm and	Descript ion
Find	Finds the specified text.
FindBack	Finds the previous occurrence of the specified text.
FindForward	Finds the next occurrence of the specified text.
FindNext	Continues the search forward, finding the next

occurrence of the specified text.

FindNextWord Finds the next occurrence of the selected text.

FindPrevious Finds the search backward, finding the previous occurrence of the specified text.

FindPreviousWord Finds the previous occurrence of the selected text.

FindRegularExpressions Searches forward for a string using [regular expressions](#).

FindRegularExpressionsBackward Searches backward for a string using [regular expressions](#).

FindReplace Continues the previous search.

FindReplaceSpecific Replaces specific text with different text.

FindReplace Replaces

placeR specific
E text with
different
text
found by
using
[regular
express
ions.](#)

FindTo Activates
ol the Find
[combo
box](#) tool.

Increm Starts an
entalSe incremen
arch tal
search
forward.

Increm Starts an
entalSe incremen
archBa tal
ck search
backwar
d.

Increm Starts a
entalSe regular
archRE expressi
on
incremen
tal
search
forward.

Increm Starts a
entalSe regular
archRE expressi
Back on
incremen
tal
search
backwar
d.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Finding Text in a Single File

With the Find command, you can search the active window for the following types of text strings:

- Whole Word Match Matches all occurrences of a text string not preceded or followed by an alphanumeric character or the underscore (_).
- Case Match Searches for text that matches the capitalization of the text string.
- Regular Expressions Uses special character sequences—regular expressions—to search for text. If you select the Regular Expression check box in the Find dialog box, you can build the search string using regular expressions from the drop-down list.

You can set bookmarks at every occurrence of the text string or expression. You can then use the Next Bookmark command to move to each bookmark in your file.

{ewl msdncd, EWGraphic, jug14c 0 /a "build.bmp"} To find a text string

- 1 Move the insertion point to where you want to begin your search.
The editor uses the location of the insertion point to select a default search string.
- 2 From the Edit menu, choose Find.
The Find dialog box appears.
- 3 In the Find What text box, type the search text or a regular expression.
—or—
Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. If you do use regular expressions, be sure the Regular Expression check box is selected.
You can also use the drop-down list to select from a list of up to 16 previous search strings.
- 4 Select any of the Find options.
- 5 To begin your search, choose Find Next or Mark All. The Find dialog box disappears when the search begins. To repeat a find operation, use the shortcut keys or toolbar buttons.
- 6 To continue your search, use the Find Next or Find Previous toolbar buttons.

{ewl msdncd, EWGraphic, jug14c 1 /a "build.bmp"} To begin a find without the Find dialog box

- 1 Type or select a search string in the Standard toolbar Find box.
- 2 Press ENTER.

Note You can use regular expressions with the Standard toolbar Find box if you have previously selected the Regular Expression check box in the Find dialog box.

{ewl msdncd, EWGraphic, jug14c 2 /a "build.bmp"} To find a string using incremental search

- 1 Press the IncrementalSearch key combination.
The status bar displays the task to be performed.
- 2 Begin typing the search string.
As you type each character, the text editor selects the matching string in your file and the status

bar displays the search string.

- 3 If necessary, press the IncrementalSearch key combination to go to the next match in your file.
- 4 Press the ESC key or use any of the navigational commands to end the search.

Note If there is no match, the text editor beeps and displays a warning in the status bar.

Tip You can assign shortcut keys to three of the options in the Find dialog box: EditToggleCaseSensitivity, EditToggleFindMatchWord, and EditToggleRE. By using the shortcut keys, you can change the search criteria without displaying the Find dialog box.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Finding Text in Multiple Files

With the Find in Files command on the File menu, you can search multiple text files for the following types of text strings:

- **Whole Word Match** Matches all occurrences of a text string not preceded or followed by an alphanumeric character or an underscore (`_`).
- **Case Match** Searches for text that matches the capitalization of the text string.
- **Regular Expressions** Uses special character sequences—regular expressions—to search for text. If you select the Regular Expression check box in the Find dialog box, you can build the search string using regular expressions from the drop-down list.

{ewl msdncd, EWGraphic, jug15c 0 /a "build.bmp"} To find a text string in multiple source files

- 1 From the File menu, choose Find In Files.

The Find In Files dialog box appears.

- 2 In the Find What text box, type the search text or a regular expression.

—or—

Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. If you do use regular expressions, be sure the Regular Expression check box is selected.

You can also use the drop-down list to select from a list of up to 16 previous search strings.

- 3 In the In Files Of Type box, select the file types you want to search.

You can use the drop-down list to select from common file types or to type text specifying other file types.

- 4 In the In Folder box, select the primary folder that you want to search. Choose the Browse button (...) to display the Choose Directory dialog box if you want to change drives and directories.

- 5 If necessary, select one or more of the Find options.

- 6 To select additional folders to search, choose the Advanced button.

The Look In Additional Folders portion of the dialog box appears.

- 7 If necessary, select the Look In Folders For Project Source Files check box.

- 8 If necessary, select the Look In Folders For Project Include Files check box.

Note These project source and project include file folders are the same as the project's directory paths. For more information on how to view and change these directory paths, see Setting Directories.

- 9 To add a folder to the Look In Additional Folders list, double-click the empty selection. Then type the path and filename, or use the Browse button (...) to display the Choose Directory dialog box to change drives and directories.

To remove a folder from the Look In Additional Folders list, select the folder and press DEL.

Developer Studio retains the contents of the Find In Files list between uses of the Find In Files command in any single session.

- 10 Choose the Find button to begin the search.

The Output window displays the list of file locations where the text string appears. Each

occurrence lists the fully qualified filename, followed by the line number of the occurrence and the line containing the match.

- 11** To open a file containing a match, double-click the entry in the Output window.

An editor window containing the file opens, with the line containing the match selected. You can jump to other occurrences of the text string by double-clicking the specific entries in the Output window, or you can use the GoToNextErrorTag command.

When you jump to a found string location specified in the Output window, the corresponding source file is loaded if it is not already open in the editor.

Note The Output window is a virtual window that is maintained even when it is not displayed. You can display the output from your last multiple-file search done during your current session by choosing the Output command from the View menu and by choosing the Find In Files tab in the Output window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Replacing Text

With the Replace command, you can search the active window for the following types of text strings, and replace each with another text string:

- **Whole Word Match** Matches all occurrences of a text string not preceded or followed by an alphanumeric character or an underscore (_).
- **Case Match** Searches for text that matches the capitalization of the text string.
- **Regular Expressions** Uses special character sequences—regular expressions—to search for text. If you select the Regular Expression check box in the Find dialog box, you can build the search string using regular expressions from the drop-down list.

{ewl msdncd, EWGraphic, jug16c 0 /a "build.bmp"} To replace text

- 1 Move the insertion point to where you want to begin your search.
The editor uses the location of the insertion point to select a default search string.
- 2 From the Edit menu, choose Replace.
The Replace dialog box appears.
- 3 In the Find What text box, type the search text or a regular expression.

Tip Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. You can also use the drop-down list to select from a list of up to 16 previous search strings. If you do use regular expressions, be sure the Regular Expression check box is selected.

- 4 In the Replace With text box, type the replacement text.
Select the menu button to the right of the combo box to display a list of replacement options.
- 5 Select any of the remaining Find options.
- 6 To begin the search, choose the Find Next button.
The Replace command selects the first matching text string.
- 7 Replace the current selection by choosing the Replace button.
—or—
Replace all identical strings by choosing the Replace All button.
—or—
Skip the current selection and find the next selection by choosing the Find Next button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Regular Expressions with Developer Studio

A regular expression is a search string that uses special characters to match a text pattern in a file. You can use regular expressions with both the Find and Replace commands.

{ewl msdn cd, EWGraphic, jug17c 0 /a "build.bmp"} To use a regular expression

- 1 From the Edit menu, choose either Find or Replace.
- 2 In the Find What text box, type a regular expression.
- 3 In the Replace With text box, type a regular expression if required.

Tip Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. You can also use the drop-down list to select from a list of up to 16 previous search strings. If you do use regular expressions, be sure the Regular Expression check box is selected.

The following table lists valid regular expressions.

Regular Expression	Description
.	Any single character.
[]	Any one of the characters contained in the brackets, or any of an ASCII range of characters separated by a hyphen (-). For example, b[aeiou]d matches bad, bed, bid, bod, and bud, and r[eo]+d matches red, rod, reed, and rood, but not reod or roed.
x[0-9]	matches x0, x1, x2, and so on.

If the first character in the brackets is a caret (^), then the regular expression matches any characters except those in the brackets.

^ The beginning of a line.

\$ The end of a line.

\ (Indicates a tagged expression to retain for replacement purposes. If the expression in the Find What text box is \ (lpsz\)BigPointer, and the expression in the Replace With box is \1NewPointer, all selected occurrences of lpszBigPointer are replaced with lpszNewPointer.

Each occurrence of a tagged expression is numbered according to its order in the Find What text box, and its replacement expression is \n, where 1 corresponds to the first tagged expression, 2 to the second, and so on. You can have up to nine

tagged
expressions.

\~ Not the
following
character. For
example,
b\~ad matches
bbd, bcd, bdd,
and so on, but
not bad.

\ Any one of the
{c characters
\! separated by
c\ the alternation
} symbol (\!).

For example, \
{j\!u\}
+fruit finds
jfruit,
jjfruit,
ufruit,
ujfruit,
uufruit, and
so on.

* None or more of
the preceding
characters or
expressions.
For example,
ba*c matches
bc, bac, baac,
baaac, and so
on.

+ At least one or
more of the
preceding
characters or
expressions.
For example,
ba+c matches
bac, baac,
baaac, but not
bc.

\ Any sequence
{\ of characters
} between the
escaped
braces. For
example, \
{ju\}+fruit
finds jufruit,
jujufruit,
jujujufruit,

and so on. Note that it will not find jfruit, ufruit, or ujfruit, because the sequence ju is not in any of those strings.

[^ Any character
] except those following the caret (^) character in the brackets, or any of an [ASCII](#) range of characters separated by a hyphen (-). For example, x[^0-9] matches xa, xb, xc, and so on, but not x0, x1, x2, and so on.

\ : Any single
a alphanumeric character [a-zA-Z0-9].

\ : Any white-
b space character. The \ :b finds tabs and spaces. There is no alternate syntax to express :b.

\ : Any single
c alphabetic character [a-zA-Z].

\ : Any decimal
d digit [0-9].

\ : Any unsigned
n number \ { [0-9]+ \ . [0-9]* \ ! [0-9]* \ . [0-9]+ \ ! [0-9]+ \ } .
For example, \ :n should

match 123, .45,
and 123.45.

\: Any unsigned
z decimal integer
[0-9]+.

\: Any
h [hexadecimal](#)
number [0-
9a-fA-F]+.

\: Any C/C++
i identifier [a-
zA-Z_ \$] [a-
zA-Z0-9_ \$]+.

\: Any English
w word (that is, a
string of
alphabetic
characters) [a-
zA-Z]+.

\: Any quoted
q string \ { \" [^ \"] * \" ! ' [^ '] * ' \ }.

\ Removes the
pattern match
characteristic in
the Find What
text box from
the special
characters
listed above.
For example,
100\$ matches
100 at the end
of a line, but
100\\$ matches
the character
string 100\$
anywhere on a
line.

Note You can use regular expressions with the Find button on the toolbar if you have previously selected the Regular Expression check box in the Find dialog box or the Replace dialog box.

```
{ewl msdncl.dll, ewcright, /c"Microsoft"}
```


Using Regular Expressions with BRIEF Emulation

A regular expression is a search string that uses special characters to match a text pattern in a file. You can use regular expressions with both the Find and Replace commands.

{ewl msdn cd, EWGraphic, jug18c 0 /a "build.bmp"} To use a regular expression

- 1 From the Edit menu, choose either Find or Replace.
- 2 In the Find What text box, type a regular expression.
- 3 In the Replace With text box, type a regular expression, if required.

Tip Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. You can also use the drop-down list to select from a list of up to 16 previous search strings. If you do use regular expressions, be sure the Regular Expression check box is selected.

The following table lists valid regular expressions for the BRIEF emulation.

Reg ular Exp res sio n	Description
?	Any single character.
[]	Any one of the characters contained in the brackets, or any ASCII range of characters separated by a hyphen (-). For example, b[aeiou]d matches bad, bed, bid, bod and bud, and r[eo]+d matches red, rod, reed and rood, but not reod or roed. x[0-9] matches x0, x1, x2, and so on. If the first

character in the brackets is a tilde (~), then the regular expression matches any characters except those in the brackets.

% The beginning of a line.

\$ The end of a line.

{ } Indicates a tagged expression to retain for replacement purposes. If the expression in the Find What text box is {lpsz}BigPointer, and the expression in the Replace With box is \0NewPointer, all selected occurrences of lpszBigPointer are replaced with lpszNewPointer.

Each occurrence of a tagged expression is numbered according to its order in the Find What text box, and its replacement expression is \n, where 0 corresponds to the first tagged expression, 1

to the second,
and so on.
You can have
up to ten
tagged
expressions.

~ Not the
following
character. For
example,
b~ad matches
bbd, bcd,
bdd, and so
on, but not
bad.

{c|
c} Any one of the
characters
separated by
the alternation
symbol (|).
For example,
{j|u}+fruit
finds jfruit,
jjfruit,
ufruit,
ujfruit,
uufruit, and
so on.

@ None or more
of the
preceding
characters or
expressions.
For example,
ba@c matches
bc, bac,
baac, baaac,
and so on.

+ At least one or
more of the
preceding
characters or
expressions.
For example,
ba+c matches
bac, baac,
and baaac,
but not bc.

[] Any sequence
of characters
between the
brackets. For
example,

[ju]+fruit
finds
jufruit,
jujufruit,
jujujufruit
, and so on.
Note that it will
not find
jfruit,
ufruit, or
ujfruit
because the
sequence ju
is not in any of
those strings.

[~] Any character
except those
following the
tilde character
(~) in the
brackets, or
any of an
[ASCII](#) range
of characters
separated by
a hyphen (-).
For example,
x[~0-9]
matches xa,
xb, xc, and so
on, but not x0,
x1, x2, and so
on.

[a-zA-Z0-9] Any single
alphanumeric
character.

[\x09] Any white-
space
character.

[a-zA-Z] Any single
alphabetic
character.

[0-9] Any decimal
digit.

[0-9a-fA-F] Any
[hexadecima](#)
l number.

{[0-9]+} Any unsigned
number. For
example,
. { [0-9]+ } . [0-

[0-9]@}|{[0-9]@ 9]@.[0-9]| 9]+}|{[0-9]+} should
 - match 123,
 9]@ .45, and
 . 123.45.
 [0-9]+
 }|
 {[0-
 -
 9]+
 }

[0- Any unsigned
 9]+ decimal
 integer.

[a- C/C++
 zA- identifier.

Z_ \$
]

[a-
 zA-
 Z
 0-
 9_ \$
]@

[a- Any English
 zA- word (that is,
 Z]+ any string of
 alphabetic
 characters).

"[~ Any quoted
 "]"@ string.
 "

\ Removes the
 pattern match
 characteristic
 in the Find
 What text box
 from the
 special
 characters
 listed above.
 For example,
 100\$ matches
 100 at the end
 of a line, but
 100\\$
 matches the
 character
 string 100\$
 anywhere on a
 line.

Note You can use regular expressions with the Find button on the toolbar if you have previously selected the Regular Expression check box in the Find dialog box or the Replace dialog box.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Regular Expressions with Epsilon Emulation

A regular expression is a search string that uses special characters to match a text pattern in a file. You can use regular expressions with both the Find and Replace commands.

{ewl msdn cd, EWGraphic, jug19c 0 /a "build.bmp"} To use a regular expression

- 1 From the Edit menu, choose either Find or Replace.
- 2 In the Find What text box, type a regular expression.
- 3 In the Replace With text box, type a regular expression if required.

Tip Select the menu button to the right of the combo box to display a list of regular search expressions. When you select an expression from this list, the expression is substituted as the search text in the Find What text box. You can also use the drop-down list to select from a list of up to 16 previous search strings. If you do use regular expressions, be sure the Regular Expression check box is selected.

The following table lists valid regular expressions for the Epsilon emulation.

Reg ular Exp res sio n	Description
.	Any single character.
[]	Any one of the characters contained in the brackets or any of an ASCII range of characters separated by a hyphen (-). For example, b(aeiou)d matches bad, bed, bid, bod and bud, and r(eo)+d matches red, rod, reed and rood, but not reod or roed. x(0-9) matches x0, x1, x2, and so on.
	If the first

character in the brackets is a caret (^), then the regular expression matches any characters except those in the brackets.

^ The beginning of a line.

\$ The end of a line.

() Indicates a tagged expression to retain for replacement purposes. If the expression in the Find What text box is (lpSz)BigPointer, and the expression in the Replace With box is #1NewPointer, all selected occurrences of lpSzBigPointer are replaced with lpSzNewPointer.

Each occurrence of a tagged expression is numbered according to its order in the Find What text box, and its replacement expression is #n, where 1 corresponds to the first tagged expression, 2

to the second,
and so on.
You can have
up to nine
tagged
expressions.

~ Not the
following
character. For
example,
b~ad matches
bbd, bcd,
bdd, and so
on, but not
bad.

(c| Any one of the
c) characters
separated by
the alternation
symbol (|).
For example,
(j|u)+fruit
finds jfruit,
jjfruit,
ufruit,
ujfruit,
uufruit, and
so on.

* None or more
of the
preceding
characters or
expressions.
For example,
ba*c matches
bc, bac,
baac, baaac,
and so on.

+ At least one or
more of the
preceding
characters or
expressions.
For example,
ba+c matches
bac, baac,
and baaac,
but not bc.

[^] Any character
except those
following the
caret (^) in the
brackets, or

any of an
[ASCII](#) range
of characters
separated by
a hyphen (-).
For example,
`x[^0-9]`
matches `xa`,
`xb`, `xc`, and so
on, but not `x0`,
`x1`, `x2`, and so
on.

[a-
zA-
Z0-
9]
Any single
alphanumeric
character.

[<t
ab>
]+
Any white-
space
character.

[a-
zA-
Z]
Any single
alphabetic
character.

{0-
9}
Any decimal
digit.

[0-
9a-
fA-
F]+
Any
[hexadecima](#)
l number.

([0
-
9]+
.
[0-
9]*
|
[0-
9]+
|
[0-
9]*
.
[0-
9]+
|
[0-
9]+
)
Any unsigned
number. For
example,
([0-9]+.
[0-9]*|
[0-9]+
|
[0-9]*.
[0-9]+
|
[0-9]+)
should
match 123,
.45, and
123.45.

[0-
9]+
Any unsigned
decimal
integer.

[a-
zA-
Z_-\$
]
C/C++
identifier.

[a-
zA-
Z

0-
 9_\$
]*
 [a-zA-Z]+ Any English word (that is, any string of alphabetic characters).
 "[~"]* Any quoted string.
 \"
 \ Removes the pattern match characteristic in the Find What text box from the special characters listed above. For example, 100\$ matches 100 at the end of a line, but 100\\$ matches the character string 100\$ anywhere on a line.

Note You can use regular expressions with a search in a single source file with the Find button on the toolbar if you have previously selected Regular Expression in the Find or Replace dialog box.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Selecting Text

You can select lines, multiple lines, and column blocks of text to cut, copy, delete, indent, and unindent. Most of the selection commands have extensions (the word “Extend” is appended to the name of the command) that move the cursor and extend the selection. By default, these commands are bound to the same key combination as the primary selection command plus the SHIFT key (such as SHIFT+LEFT ARROW for CharLeftExtend).

{ewl msdn cd, EWGraphic, jug20c 0 /a "build.bmp"} To select a line of text

- In the selection margin, point to the beginning of the text you want to select and click the left mouse button.

{ewl msdn cd, EWGraphic, jug20c 1 /a "build.bmp"} To select multiple lines of text

- 1 In the selection margin, point to the beginning of the text you want to select.
- 2 Drag either up or down to select the lines of text.

{ewl msdn cd, EWGraphic, jug20c 2 /a "build.bmp"} To select a column block of text

- 1 Point to the beginning of the text you want to select.
- 2 Hold down the ALT key, hold down the left mouse button, and drag the mouse until the desired text has been highlighted.
- 3 When you have finished selecting the text, release the ALT key and the left mouse button.
When you release the left mouse button, the block of text is selected, and the text is available for cut, copy, delete, and indent operations. To cancel column-select mode, click the left mouse button.

Note When you use proportional fonts in the editor window, the column positions you select in the first line may not correspond exactly to the subsequent lines you select. The text editor selects the character most directly in line with the start and end columns, ignoring the actual character count.

The text editor commands for selection are described in the following table.

Comm and	Descript ion
CharLeftExtend	Extends the selection one character to the left.
CharRightExtend	Extends the selection one character to the right.
DocumentEndExtend	Extends the selection

to the
end of
the file.

Docum Extends
entStar the
tExtend selection
to the
beginnin
g of the
file.

HomeE Extends
xtend the
selection
alternatel
y
between
the start
of the
current
line and
the start
of the
text on
that line.

LineDo Extends
wnExte the
nd selection
one line
downwar
d.

LineEn Extends
dExten the
d selection
to the
end of
the text
on the
current
line.

LineUp Extends
Extend the
selection
one line
upward.

PageD Extends
ownExt the
end selection
one page
downwar
d.

PageU Extends
pExten the
d selection
one page

upward.

SelectA Selects
ll the entire
[docum
ent](#).

Select Char Starts
the character
-
selection
mode.
While
this
mode is
active, all
other
navigatio
n
comman
ds will
select
the
character
s from
the
position
where
the
comman
d was
executed
to the
current
cursor
location.

SelectL Starts
ine the line-
selection
mode.
While
this
mode is
active, all
other
navigatio
n
comman
ds will
select
lines
from the
position
where
the
comman

d was
executed
to the
current
cursor
location.

Select Starts
Column the
column-
select
mode. In
column-
select
mode,
the
navigatio
n keys
act as if
virtual
space is
enabled.

WordL Extends
eftExte the
nd selection
backward
one
word.

WordRi Extends
ghtExte the
nd selection
forward
one
word.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Editing with the Text Editor

With the text editor, you can cut, copy, and paste text using menu commands or drag-and-drop. You can also undo and redo selected editing actions.

The text editor provides the following editing commands:

- Cutting, copying, pasting, and deleting text
- Undoing and redoing editing actions
- Using drag-and-drop
- Specifying column blocks for editing

All editing commands require a selection in order to work. Some of the commands can make a selection based on the current cursor location. Command names that begin with an object (such as WordCapitalize) assume that object for a default selection; otherwise, the default selection will be the character adjacent to the cursor. For example, the Delete command removes the character to the right of the cursor if there is no selection.

Note You can enable the copy command to work on the current line even if there is no selection. From the Tools menu, select Options. Select the Compatibility tab, and select the Enable Copy Without Selection option. This enables the copy command to work on the current line if there is no selection.

When you cut text from the file, the text is removed from your file and placed on the Clipboard. When you delete text from the file, the text is removed from your file, and the Clipboard is not used. All Windows applications share one single Clipboard. Commands that use the Clipboard will overwrite whatever was previously placed onto the Clipboard by other commands or other Windows applications. This single-Clipboard behavior is true even when the IDE is emulating an editor, such as Epsilon, that supports multiple Clipboards.

The text editor commands for editing are described in the following table.

Comm and	Descript ion
CharTransposition	Swaps characters around the cursor.
Copy	Copies the selection to the Clipboard.
Cut	Removes the selection and copies it to the Clipboard.

Delete Deletes the selection.

Delete Back Deletes the selection, or if there is no selection, deletes the character to the left of the cursor.

Delete BlankLines Deletes the blank lines adjacent to the cursor.

Delete HorizontalSpace Deletes the spaces and tabs around the cursor.

Format Selection Formats the selection using the smart indent settings.

IndentSelection Indents the selected text right one tab stop.

IndentSelectionToPrevious Indents the selection to line up with the previous line's indentation.

LevelCutToEnd Cuts the text between the

cursor
and the
end of
the next
brackete
d level.

LevelC Cuts the
utToSta text
rt between
the
cursor
and the
beginnin
g of the
previous
brackete
d level.

LineCut Deletes
the
selected
lines and
places
them on
the
Clipboar
d.

LineDel Deleted
ete the
selected
line.

LineDel Deletes
eteToE to the
nd end of
the
current
line.

LineDel Deletes
eteToSt to the
art beginnin
g of the
current
line.

LineOp Opens a
enAbov new line
e above
the
cursor.

LineOp Opens a
enBelo new line
w below
the
cursor.

LineTra Swaps
nspose the

current
and
previous
lines.

LowerC
aseSel
ection Makes
the
selection
all
lowercas
e.

Paste Inserts
the
Clipboar
d
contents
at the
cursor.

Senten
ceCut Deletes
the
remainde
r of the
sentence
.

TabifyS
election Replaces
spaces
with tabs
in the
selection.

Uninde
ntSelec
tion Indents
the
selected
text left
one tab
stop.

Untabif
ySelect Replaces
tabs with
spaces in
the
selection.

UpperC
aseSel
ection Makes
the
selection
all
uppercas
e.

WordC
apitaliz
e Makes
the first
character
uppercas
e.

WordD
eleteTo
End Deletes a
word to
the right.

WordD Deletes a
eleteTo word to
Start the left.

WordL Makes
owerCa the
se current
word
lowercas
e.

WordTr Swaps
anspos the
e current
and
previous
words.

WordU Makes
pperCa the
se current
word
uppercas
e.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Cutting, Copying, Pasting, and Deleting Text

You can edit your text using the following actions.

Action	Description
Cut	Removes selected text from the active window .
Copy	Duplicates selected text in the active window .
Paste	Pastes cut or copied text into an active window .
Delete	Deletes text without copying it to the Clipboard.
Undo	Restores the text.
Redo	Re-applies the prior edit.

{ewl msdn cd, EWGraphic, jug22c 0 /a "build.bmp"} To cut or copy and paste text

- 1 Select the text you want to cut or copy.
- 2 From the Edit menu, choose Cut or Copy.
The cut or copied text is placed onto the Clipboard and is available for pasting.
- 3 Move the insertion point to any source window where you want to insert the text.

- 4 From the Edit menu, choose Paste.

{ewl msdncd, EWGraphic, jug22c 1 /a "build.bmp"} To delete text

- 1 Select the text you want to delete.
- 2 From the Edit menu, choose Delete.

The deleted text is not placed onto the Clipboard, and cannot be pasted.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Undoing and Redoing Editing Actions

Use the Undo command to undo previous editing actions. Use the Redo command to reapply editing actions that have been undone. Redo is unavailable unless you have used the Undo command.

The number and scope of editing actions you can undo is determined by the size of the text editor's UndoRedoSize buffer in the registry. For information on how to modify the Registry, see Initializing and Configuring Microsoft Developer Studio.

{ewl msdncd, EWGraphic, jug23c 0 /a "build.bmp"} To Undo an editing action

- From the Edit menu, choose Undo.
—or—

{ewl msdncd, EWGraphic, jug23c 1 /a "build.bmp"} To Redo an Undo action

- From the Edit menu, choose Redo.
—or—

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Drag-and-Drop

Drag-and-drop editing is the easiest way to move or copy a selection of text in a file or between files.

{ewl msdncd, EWGraphic, jug24c 0 /a "build.bmp"} To move text using drag-and-drop editing

- 1 Select the text you want to move.
- 2 Drag the selected text to the new location.

Note You can also use the right mouse button for drag-and-drop editing. Select the text you want, and then use the right mouse button to drag the text to a new location. A pop-up menu appears, asking if you want to move or copy the selected text.

Tip At any time during a drag-and-drop, you can click the other mouse button to cancel the operation.

{ewl msdncd, EWGraphic, jug24c 1 /a "build.bmp"} To copy text using drag-and-drop

- 1 Select the text you want to copy.
- 2 While holding down the CTRL key, drag the selected text to the new location.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Recording and Playing Back Keystrokes

With the text editor, you can automate keyboard tasks by recording and playing back keystrokes. The playback feature is available until you record a new set of keystrokes or end the editing session.

You can play back recorded keystrokes only into a single editor view. If you activate a new editor view while recording keystrokes, the recorder will remain in record mode and continue recording keystrokes. However, when the recorded keystrokes are played back, they will be played back into the view they were recorded from. If a window is closed, playback stops.

Note During recording, all mouse-driven selections in text windows are disabled.

{ewl msdncd, EWGraphic, jug25c 0 /a "build.bmp"} To record keystrokes

- 1 Move the mouse pointer to where you want to begin typing.
- 2 From the Tools menu, choose Record Keystrokes.
The Record toolbar appears.
- 3 Record the keystrokes that you want.
During recording, all mouse-driven selections are disabled. Keystrokes are entered at the location you have selected.
- 4 From the Tools menu, choose Stop Recording when you have finished recording your keystrokes.

{ewl msdncd, EWGraphic, jug25c 1 /a "build.bmp"} To play back keystrokes

- 1 Move the mouse pointer to where you want to play back the recorded keystrokes.
- 2 From the Tools menu, choose Playback Recording.
The recorded keystrokes will be played back into the active editor window at the location you have selected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Text Editor Options

With Developer Studio, you can set the text editor's behavior to suit your preferences and work habits. You can customize the text editor by:

- Setting editor emulation.
- Setting file save preferences.
- Setting and using the selection margin.
- Setting tabs and indents.
- Setting the font style, size, and color.
- Setting syntax coloring.
- Emulate two popular text editors: BRIEF and Epsilon.

The text editor commands for editor settings are described in the following table.

Comm and	Descript ion
EditTog gleCas eSensit ivity	Toggles the search case sensitivit y.
EditTog gleFind Match Word	Toggles match whole word.
EditTog gleOve rtype	Toggles between inserting and replacing typing.
EditTog gleRE	Toggles the regular expressi on search.
EditTog gleTab Display	Shows or hides the tab character s.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting Editor Emulation

The Developer Studio text editor can emulate two popular text editors: BRIEF and Epsilon. With the emulation feature, the text editor can emulate the key bindings, text selection, caret display, and window display, as well as most editing commands of the selected editor.

{ewl msdncd, EWGraphic, jug27c 0 /a "build.bmp"} To set an editor emulation

- 1** From the Tools menu, choose Options.
The Options dialog box appears.
- 2** Select the Compatibility tab.
- 3** In the Recommended Options For list box, select the editor that you want to emulate.
The default editor is Developer Studio.
The Options box displays the status of pre-defined editor options.
- 4** Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Save Preferences

You can set save preferences—such as whether to be prompted before saving a file—in the Options dialog box. As a default, the text editor saves all changed files prior to building an application. The following table lists the save preferences:

Save Option	Description
Save before running tools	Saves files before you build a project or run a build utility such as NMAKE.
Prompt before saving files	Confirms (with a dialog box prompt) that you want to save files.
Automatically reload of externally modified files	Automatically reloads externally modified files that have been loaded (but not yet changed) by the editor.

{ewl msdncd, EWGraphic, jug28c 0 /a "build.bmp"} To change the save options

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Editor tab.
- 3 Select any of the Save Options.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting and Using the Selection Margin

The selection margin is an area to the left of each line of text. You can use the mouse in this area to select text. The selection margin also displays information about source lines. Breakpoints, bookmarks, the extended instruction pointer (EIP), and the tag pointer are all indicated by icons in the selection margin.

{ewl msdncd, EWGraphic, jug29c 0 /a "build.bmp"} To set the selection margin

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Editor tab.
- 3 Select the Selection Margin check box.
- 4 Choose OK.

{ewl msdncd, EWGraphic, jug29c 1 /a "build.bmp"} To use the selection margin

- When the mouse pointer is moved into the selection margin, it changes to an up-and-right-pointing select arrow (a mirror image of the standard select arrow). Do any of the following:
 - Click in the margin to select the entire line to the right of the mouse pointer.
 - Click in the margin and move the mouse pointer to select multiple consecutive lines.
 - While holding down SHIFT, click in the margin and move the mouse pointer to extend a selection.
 - While holding down CTRL, click anywhere in the margin to select the entire file. (This is equivalent to choosing the Select All command from the Edit menu).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Tabs and Indents

You can indent text with tab characters several ways:

- Use the TAB key.
- Use Auto Indent (without Smart Indent)
- Use Auto Indent (with Smart Indent enabled)

When you press the TAB key, the insertion point moves to the next indent level. You can display (or hide) the tab symbols by pressing the EditToggleTabDisplay key combination.

Note You can display all of the current keyboard shortcuts. For more information, see Displaying the Keyboard Shortcuts.

You can also use Auto Indent (without Smart Indent) to automatically indent new lines to match the previous line.

{ewl msdn cd, EWGraphic, jug30c 0 /a "build.bmp"} To set Auto Indent

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Tabs tab.
- 3 Under Auto Indent, select the appropriate setting.
- 4 Choose OK.

If you use Auto Indent (with Smart Indent enabled), the text editor automatically indents the text based on the context of the previous lines.

{ewl msdn cd, EWGraphic, jug30c 1 /a "build.bmp"} To set Smart Indent

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Tabs tab.
- 3 Under Auto Indent, select the Smart option.
- 4 Under Smart Indent Options, select the language element and specify the number of previous lines to use for the context of smart indenting.
- 5 Choose OK.

Backspacing over a tab character deletes the tab character, regardless of the indent setting.

If you select the Insert Spaces option, a tab character is not inserted, and only spaces are inserted to reach the next indent level.

Note The File Type list box on the Tabs tab contains a list of file types. The initial tab settings for each file you load are assigned based on the file extension and the setting of this list box. You can use the Tab Size box and Indent Size box on the Tabs tab or on the Source Window property page to specify individual settings for these two fields on a per-file basis, as needed.

{ewl msdn cd, EWGraphic, jug30c 2 /a "build.bmp"} To change tab and indent settings

- 1 From the Tools menu, choose Options.

The Options dialog box appears.

- 2 Select the Tabs tab.
- 3 In the Tab Size box, type the number of spaces to use as a tab stop. The default is four spaces.
- 4 In the Indent Size box, type the number of spaces to use for indents. The default is four spaces.
- 5 Select the Keep Tabs option to treat each tab as a single tab character when the file is saved.
—or—
Select the Insert Spaces option to use spaces as specified in the Tab Size box.
- 6 Choose OK.

{ewl msdncd, EWGraphic, jug30c 3 /a "build.bmp"} To change tab and insert settings using the Source Window property page

- 1 Click the right mouse button with the mouse pointer in the source window,
- 2 From the pop-up menu, choose Properties.
The Source Window property page appears.
- 3 In the Tab Size box and Indent Size box, type the tab and indent setting.

{ewl msdncd, EWGraphic, jug30c 4 /a "build.bmp"} To display or hide tab symbols

- Press the EditToggleTabDisplay key combination to toggle the display of tab symbols. Tab symbols are displayed as >> whenever there is a tab in a source file.

You can press the TAB key to move the caret to the next indent level. You can also move a block of lines one tab to the right or left.

{ewl msdncd, EWGraphic, jug30c 5 /a "build.bmp"} To indent a group of lines

- 1 Select the group of lines.
- 2 Press the IndentSelection key combination.

{ewl msdncd, EWGraphic, jug30c 6 /a "build.bmp"} To unindent a group of lines

- 1 Select the group of lines.
- 2 Press the UnindentSelection key combination.

Note The UnindentSelection command only returns to previous tab stops.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting the Font Style, Size, and Color

You can change the font style, size, and color settings for any window within Developer Studio with the Format command. You may find different fonts in various windows give visual clues about the function of the windows—the default setting for source windows, a different font for the Watch window, and so on. You can use the text font and size to better manage your window display of information.

Note In addition to setting font coloring, you must enable syntax coloring in order to view colored text elements.

{ewl msdncd, EWGraphic, jug31c 0 /a "build.bmp"} To change a font style, size, or color

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Format tab.
- 3 In the Category box, select the category of information to be formatted.
The Category list box displays the windows that have formatting options.
- 4 In the Font box, select the font to be used for the category you selected.
The Font drop-down list box displays the different fonts installed on your system. The text sample in the sample box changes to the font you select.
- 5 In the Size box, select the Size to be used for the font you selected.
The Size drop-down list displays the sizes available for the selected font. The text sample in the sample box changes to the size you select.
- 6 In the Colors list box, select the type of text you want to color.
- 7 In the Background list box, select a background color.
- 8 In the Foreground list box, select a foreground color.

Note The Background and Foreground lists display the 16 standard colors and the Automatic setting. The text sample displayed in the Sample box changes to the color you select.

The behavior of the Automatic color setting depends on the element selected. For colors that map to standard system elements (such as Foreground color, Background color, or Text Selection color), the Automatic setting sets the element to the appropriate system color. For syntax coloring elements and other non-system defined colors, the Automatic setting indicates that the foreground color or background color from the same category is to be used.

- 9 Choose OK.

Text within one category of window can be only one font and size. Multiple fonts cannot be displayed in the same category of source window.

The font and size settings apply to everything within the selected category, while the foreground and background color settings apply only to the selected element of that category.

Tip You can reset the formatting options for a selected Category to the default settings by choosing Reset.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Syntax Coloring

Using different colors for various elements of your display, such as keywords, identifiers, or strings, gives you visual cues about the structure of your source code. These changes are global and affect all source files with extensions recognized by the installed language.

{ewl msdncd, EWGraphic, jug32c 0 /a "build.bmp"} To set syntax coloring in an individual source file

- 1 Click the source file window or use the Window menu to make the source window active.
If there are multiple windows open on the source file, select one. Syntax coloring changes will appear in all windows opened on the source file.
- 2 From the Edit menu, choose Properties.
The Source Window property page appears. The Language list box displays the current language setting for syntax coloring. The drop-down list contains the installed language choices.
Note The Source Window property page may also be accessed by clicking the right mouse button in the source window.
- 3 In the Language list box, select Java to set syntax coloring for that source file, or select None to turn syntax coloring off.

Note Global syntax coloring for Java is enabled by default.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

HTML Syntax Coloring for Source Files

Visual J++ version 1.0 supports HTML syntax coloring for source files. Software that support HTML differ widely in the set of elements, attributes, and entities they recognize. The HTML syntax coloring feature of Visual J++ version 1.0 lets you select a range of HTML support, and allows custom variations. When you open an HTML file, the default HTML support is Microsoft Internet Explorer 3.0. To switch to a different support, open the Properties Page for this source file. You can choose Properties from the Edit menu or press ALT+ENTER. Select the HTML support of your choice from the Language list box. The HTML support you have selected becomes the new default.

The built-in HTML support includes:

- HTML 2.0. This conforms with the DTD given in RFC1866.
- Microsoft Internet Explorer 2.0.
- Microsoft Internet Explorer 3.0.

Here is what you do to define a custom variation. Place a text file with extension .HLX in the \MSDEV\BIN\IDE directory. The first line of the file is a signature that uniquely identifies the type of the file. It contains the name of the variation that will be displayed on the Properties Page of the source file. After the signature line, the format resembles that of a Windows initialization file. A semicolon at the beginning of a line indicates a comment.

The file should contain three sections, the set of elements, attributes, and entities that will be colored. Each section contains a list of names separated by space or carriage return/line feed. The names do not need to be in alphabetical order. You must specify all the elements, attributes, and entities you want to color.

Note If the custom variation you specify in the .HLX file has the same name as a built-in Visual J++ HTML support, it will override the built-in HTML support. There is a 14-character limit to the name of the HTML variation you specify.

The following examples gives the general outline of an .HLX file. For brevity, this example is shortened to omit most names.

```
@HLX@ "HTML - Custom"
; Custom HTML tagset file.
; Must begin with the "@HLX@" signature and
; the name of the HTML variant in quotes. The
; name is limited to 14 characters.
;-----
[Elements]
; Element set, case-insensitive
; HTML tags, e.g. <H1>
A APPLET
; break control
BR NOBR WBR
; headings
H1 H2 H3 H4 H5 H6
; character formatting
B CITE CODE EM I KBD STRIKE STRONG TT U VAR
FONT S SUB SUP SMALL BIG BLINK DFN; etc. ...

;-----
[Attributes]
; Attribute set, case-insensitive
; (applied to all elements)
ACTION ALIGN ALINK ALT
```



```
; etc. ...
```

```
;-----
```

```
[Entities]
```

```
; Entity set, case-sensitive
```

```
; special characters, e.g. &amp;
```

```
; If this section is empty or omitted, the standard set from RFC1866 is  
used.
```

```
gt lt amp quot
```

```
; etc. ...
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Managing Open Windows

The text editor features options that control the display of source windows. You can switch between windows, open new windows, split window views, and view a source file in full-screen mode.

{ewl msdncd, EWGraphic, jug34c 0 /a "build.bmp"} To switch to a source window

- 1 Click anywhere in the window.
—or—
From the Window menu, choose the filename.
—or—
From the Window menu, choose Windows.
The Windows dialog box appears.
- 2 Select a window from the Select Window list.
- 3 Choose the Activate button, or double-click the selection.

{ewl msdncd, EWGraphic, jug34c 1 /a "build.bmp"} To create a new window for an open source file

- 1 Switch to the source window.
- 2 From the Window menu, choose the New Window command.
A second copy of the source file is displayed with an *:n* suffix. As you open more windows on the source file, the value of *n* increases. You can scroll and split each window independently. You can make changes to the source file from any window.

Note When you first open a file, if you select the Read Only check box in the Open dialog box, the current window and any duplicates of the window remain read only.

{ewl msdncd, EWGraphic, jug34c 2 /a "build.bmp"} To split a source window

- 1 Click the split bar at the top of the vertical scroll bar, and drag it down to the location you want.
—or—
Switch to the source window.
If there are multiple windows open on the source file, select one of them.
- 2 From the Window menu, choose Split.
The split bar appears.
- 3 Drag the split bar to the location you want.

{ewl msdncd, EWGraphic, jug34c 3 /a "build.bmp"} To view a source file in full-screen mode

- 1 Switch to the source window.
- 2 From the View menu, select Full Screen.
The source window is displayed in full-screen mode. A small button appears at the top that allows you to reset the screen to regular mode.
Initially, the toolbars, status bar, and scroll bars are hidden. From the Tools menu (ALT+T), choose Options and then use the Editor tab to control window settings.

{ewl msdncd, EWGraphic, jug34c 4 /a "build.bmp"} To end full screen mode

- Press the ESC key.
–or–
Click the Full-Screen button.

All files are automatically closed when you quit Developer Studio (you will be prompted to save any changed files). You can also close any individual source file without quitting the application.

{ewl msdncd, EWGraphic, jug34c 5 /a "build.bmp"} To close a source file

- 1 From the Window menu, choose Windows.
The Windows dialog box appears.
- 2 Select one or more files from the Select Window list box.
- 3 Choose the Close Window button.
–or–
Switch to the source window.
- 4 From the File menu, choose Close to close the active window and any additional views of the window.
–or–
If the window is not maximized, double-click the window's Control-menu box. When you double-click the Control-menu box, the window is closed, but additional views of the document remain open.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Source-Code Control

Source-code control systems enable you to track changes to source-code files during the course of software development. With source-code control systems, you can ensure that changes are not overwritten in projects with multiple authors, and that authors are working with the most up-to-date code. You can also return to earlier versions of code, if necessary.

Microsoft Developer Studio provides facilities for integrating a source-code control system into the development environment. If you install a source-code control system that conforms to the Microsoft Common Source Code Control Interface, you can directly access source-code control functionality from the Developer Studio menus.

Note Until you install a source-code control system that conforms to the Microsoft Common Source Code Control Interface, the menu commands for source-code control will not appear. In addition, either you or the installation program must make the correct entries in the Registry.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting Up Source-Code Control

To use the integrated source-code control capabilities in Microsoft Developer Studio, you must take the following steps:

- Install a source-code control system that conforms to the Microsoft Common Source Code Control Interface.
- Ensure that the installation program for the source-code control system writes the correct information to the Registry, and if it doesn't, make the correct entries.
- Complete all the administrative tasks required by your source-code control system. These tasks may include designating locations for master versions of files, creating network connections, setting permissions for drives and/or directories, or adding user information and permissions to the system.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Supported Source-Code Control Functionality

Microsoft Developer Studio provides commands for a number of common source-code control operations used in everyday work. It supports the following operations:

- Putting an entire project under source-code control.
- Putting individual files under source-code control.
- Getting current versions of files.
- Checking files out of the source-code control system.
- Checking files into the source-code control system and merging others changes.
- Checking files into the source-code control system and ignoring changes.
- Removing files from the source-code control system.
- Viewing the history of changes made to a file.
- Viewing the differences between the local copy of a file and its master copy.

If your installed source-code control system supports other operations in addition to these basic ones, Microsoft Developer Studio provides access to them from the Advanced button on the relevant dialog box. These other operations could include such things as checking out files exclusively to prevent other users from working on them at the same time.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Unsupported Source-Control Functionality

Microsoft Developer Studio supports basic functionality in installed source-code control systems, as outlined in the section Supported Source-Code Control Functionality . This integrates common source-code control operations into your customary working environment.

These integrated operations do not encompass all possible capabilities of source-code control systems. For certain source-code control operations, you will have to use the programs or features of your installed source-code control system. These operations include administrative tasks, such as designating locations for master versions of files, creating network connections, setting permissions for drives and/or directories, or adding user information and permissions to the system.

In addition, if your source-code control system allows certain operations on files in the source-code control tree, such as getting specific versions of files, branching, or merging branches, you need to use the installed source-code control program for those operations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Putting Files Under Source-Code Control

When you add a file to your source-code control system, the system manages access to the file and maintains a record of all changes made to the file. It also records when a file was changed and who changed the file. From Microsoft Developer Studio, you can put entire projects under source-code control, or you can put individual files under source-code control.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Displaying the Source-Code Control Toolbar

You can access the integrated source-code control commands from buttons on a standard toolbar.

{ewl msdncd, EWGraphic, jug5d 0 /a "build.bmp"} To display the Source-Code Control toolbar

- 1 From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2 From the Toolbars list, select Source Control.
The Source-Code Control toolbar immediately appears.
- 3 Choose the Close button.

You can also remove buttons from the toolbar, add other buttons, or add source-code control buttons to other toolbars. For information on customizing toolbars, see [Working with Toolbars](#) .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding a Project to Source-Code Control

You can add a project to your source-code control system any time after you have created it.

Note Before you can add any files to source-code control, you must complete any administrative tasks required by your source-code control system, using the administrative program supplied by your system. This may include adding users or creating a source-code control project database, for instance.

{ewl msdncd, EWGraphic, jug6d 0 /a "build.bmp"} To add a project to source-code control

- 1 Open an existing project or create a new project.
- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Add To Source Control.
The installed source-code control displays one or more dialog boxes, requesting source-code control project information.
- 3 Specify the information required by the installed source-code control system.
The Add To Source Control dialog box appears, with files in the project workspace selected.
- 4 If you want to add the files to source-code control, but immediately check them out, select the Keep Checked Out check box.
- 5 In the Comment text box, type a comment about the files, if you want.

Note If your source-code control system supports additional options, an Advanced button appears on the Add To Source Control dialog box, and you can select those options at this point.

- 6 Choose the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding Individual Files to Source-Code Control

You can set Developer Studio options to prompt you automatically each time you insert files into your project, or you can explicitly choose to put them under source-code control.

{ewl msdncd, EWGraphic, jug7d 0 /a "build.bmp"} To prompt automatically for inclusion under source-code control

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Source Control tab.
- 3 Select the Prompt To Add Files When Inserted check box.
- 4 Choose the OK button.

Now, each time you insert files into the project, Developer Studio prompts you to add the inserted files to your source-code control system.

If some or all of the files currently included in your project are not under source-code control, you can add them individually to the source-code control system.

{ewl msdncd, EWGraphic, jug7d 1 /a "build.bmp"} To add individual files to source-code control

- 1 In the FileView pane of the Project Workspace window, select the files that you want to put under source-code control.
- 2 From the Tools menu, choose Source Control, and then choose Add To Source Control from the cascading menu.
The Add To Source Control dialog box appears, with checks in the Files list next to the files that you have selected. The list includes all files in the project directory that are not already under source-code control, and you may check or uncheck any files in the list.
- 3 In the Comment text box, type a comment about the files, if you want.
- 4 Choose the OK button.

The files are now under source-code control, and the file icons in the FileView pane are now grayed to indicate this.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Removing Files from Source-Code Control

When you remove files from your Microsoft Developer Studio project, you may first want to remove them from source-code control.

{ewl msdncd, EWGraphic, jug8d 0 /a "build.bmp"} To remove a file from source-code control

- 1** In the FileView pane of the Project Workspace window, select the files that you want to remove from source-code control.
- 2** From the Tools menu, choose Source Control, and from the cascading menu, choose Remove From Source Control.

The Remove From Source Control dialog box appears, with checks in the Files list next to the files that you have selected. You may check or uncheck any files in the list.

- 3** Choose OK.

Note Not all source-code control systems allow individual users to remove files from source-code control. Some systems require source-code control administrators to remove them, and the administrator may need to use the source-code control system's administrative program.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Determining the Status of Files

When you are using a source-code control system, it is important to be able to determine the status of files within the system. This can help prevent collisions in groups with multiple authors, ensure that you are working on current files, determine whether you have access to a file, and so on. You can also examine the historical status of files to determine when and what changes were made, and who made them.

You can determine some information from the FileView pane, other information from the property pages for a file, and other information from examining the files history.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Reading the FileView Pane

The FileView pane of the Project Workspace window displays all the files that are currently in the project workspace. If a project is under source-code control, the file icons are grayed, and if a file is checked out, a check mark appears to the left of the file icon. Figure 6.1 shows a file under source-code control that has been checked out.

Figure 6.1 FileView Showing a Checked Out File

```
{ewc msdncd, EWGraphic, jug10d 0 /a "uguideCHECK.BMP"}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Examining File Status on Property Pages

Each file in a project has a property page associated with it. The property page includes information about the file, including its current status in the source-code control system if it is under source-code control.

Note If the file is not under source-code control, no status information appears on the property page.

{ewl msdncd, EWGraphic, jug11d 0 /a "build.bmp"} To examine a property page from the FileView pane

- Select the file in the FileView pane of the Project Workspace window, and press ALT+ENTER.
—or—
Select the file in the FileView pane of the Project Workspace window, click the right mouse button to display the pop-up menu, and choose Properties.
—or—
Select the file in the FileView pane of the Project Workspace window, and from the Edit menu, choose Properties.

{ewl msdncd, EWGraphic, jug11d 1 /a "build.bmp"} To examine a property page from a source editor window

- Press ALT+ENTER and select the General tab.
—or—
Click the right mouse button to display the pop-up menu, choose Properties, and then select the General tab.
—or—
From the Edit menu, choose Properties, and then select the General tab.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Examining File Histories

In some cases, you may want to know what changes were made to a file, either recently or throughout its existence. You can request the source-code control system to show a history for a file or files that you have added to source-code control. Some source-code control systems allow you to select only a single file. The type of detail shown in the file histories depends on the source-code control system.

{ewl msdncd, EWGraphic, jug12d 0 /a "build.bmp"} To show file histories

- 1** In the FileView pane of the Project Workspace window, select the file or files for which you want a history.
- 2** From the Tools menu, choose Source Control, and then choose Show History from the cascading menu.

Your source-code control system displays the history for the selected file or files.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Getting Current Versions of Files

Updating your local copies of files to versions from the master source-code control files is called “getting” or “synchronizing” files. In any software project with multiple authors, you need to update your local copies frequently to ensure that you incorporate changes that other authors have made.

In a large project, changes can be made in files that you normally do not work in, but that do contain information that you use. For instance, project-wide header files may define manifest constants or macros that appear in your source files. When you get or synchronize your local files, the master versions of files are copied to your local project. The files are not checked out, and you cannot modify them and check in changes, but you can build with the most up-to-date versions.

If you have checked out files and made changes to your local copies, and other authors have made changes to those same files and checked them in, your source-code control system reports that you have changes to merge. You then need to follow the recommended procedures in your source-code control system to reconcile and verify those changes.

{ewl msdncd, EWGraphic, jug13d 0 /a "build.bmp"} To get current files in your project

- 1 In the FileView pane of the Project Workspace window, select the files that you want to get.
- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Get Latest Version.

The Get Latest Version dialog box appears, with checks next to the files that you have selected. The list includes all files in the project directory that are under source-code control, and you may check or uncheck any files in the list.

Tip You can quickly select all the items in the list by selecting the first item, pressing SHIFT+END to select all the items, and then pressing the SPACEBAR to change the check box state. If one or more files are checked, they now are unchecked. Pressing the SPACEBAR again checks them all.

- 3 Choose the OK button.

The source-code control system copies all the selected files with changes by other authors to your local directory.

You can also have Developer Studio automatically prompt you to get the current versions of files when you open a project workspace.

{ewl msdncd, EWGraphic, jug13d 1 /a "build.bmp"} To get current versions of files when opening a project workspace

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Source Control tab.
- 3 Select the Get Files When Opening The Workspace check box.
- 4 Choose the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checking Files In and Out

When you begin work on your project, normally you open the project in the project workspace and get the current versions of the project files to make sure that you are looking at the most up-to-date sources. Before you begin to modify the source files, you check them out; after you have completed the modifications, you check the files in.

When you have a file checked in, your local copy of the file is read-only, and you cannot save any changes to it. When you check out a file, you can make changes to your local copy of the file, and save those changes to the file. When you check the file in, you copy those changes to the master copy of the file in the source-code control project. This makes those changes available to your coworkers. Depending on the characteristics of your source-code control system, only one author can check out a file, or more than one author can check out a file simultaneously.

In Microsoft Developer Studio, in addition to selecting files directly from the FileView pane, you can also check them in and out by selecting from the other panes. For instance, if you select a class to check out in the ClassView pane, Developer Studio prompts you to check out files associated with that class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Checking Files Out

When you check a file out, your installed source-code control system changes the status of the file from read-only to writeable, and records that you have the file checked out. You then have the necessary permissions to revise the file. Your source-code control system may include a mechanism for exclusive use. You can then specify that you have the file checked out, and that no one else may check out that file.

Some source-code control systems allow multiple authors to check out the same file. In this case, the source-code control system merges the changes from the authors when each checks in the file.

{ewl msdncd, EWGraphic, jug15d 0 /a "build.bmp"} To check files out

- 1 In the FileView pane of the Project Workspace window, select the files that you want to check out.
- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Check Out. The Check Out File(s) dialog box appears, with checks next to the files that you have selected. The list includes all checked-in files in the project directory, and you may check or uncheck any files in the list.
- 3 Type a comment in the Comment text box, if you want.

Note Not all source-code control systems support comments when checking files out. If yours does not, this text box does not appear.

- 4 Choose the OK button.

You can also check out a file using the pop-up menu in the text editor windows. Press the right mouse button to display the menu. From the menu, choose Check Out.

You can have Developer Studio prompt you to check out a file if you start to edit it, but have not checked it out.

{ewl msdncd, EWGraphic, jug15d 1 /a "build.bmp"} To prompt for check out from editor windows

- 1 From the Tools menu, choose Options. The Options dialog box appears.
- 2 Select the Source Control tab.
- 3 Select the Check Out Source File(s) When Edited check box.
- 4 Choose the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checking Files In

When you check a file in, the source-code control system changes the status of the file from writeable to read-only, and records that you have checked in the file. It also records the differences between the contents of the file when you checked it out and when you checked it in.

You generally want to view the changes to the file before you check it in to confirm the changes that you made. In some cases, you may want to discard all changes to your file before checking it in. In other cases, you may need to merge changes that coworkers have made to the file after you checked it out. If you had the file checked out exclusively, after you check it in, others can check out the file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Viewing Your Changes to a File

It is best to review the changes that you have made in a file before you check in the file. Your source-code control system displays the differences between your local version of the file and the master version in your source-code control project.

{ewl msdncd, EWGraphic, jug17d 0 /a "build.bmp"} To view your changes

- 1 Select the file with the changes that you want to view.

Note You can select only a single file, which must already be checked out. This method reports only the differences between this version and the master version. You cannot use this method to examine differences between two files in your project, for instance.

- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Show Differences.

Your source-code control system displays the differences, or reports that the files are identical.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checking Files In and Removing Your Changes

In some cases, you may make changes to your local files, and then decide that you do not want to check the changes in to the source-code control system. You may, for instance, have viewed the local changes and discovered errors, you may have pursued some modifications that were not fruitful, or you may not have had time to completely implement some changes and do not care to check in incomplete code. In these cases, you can have the source-code control system check the files in, but ignore any changes you made.

Note If you want to save the changes before checking the files in without the changes, you can always copy the files to another location, or save them under another name using the Save As command from the File menu.

{ewl msdncd, EWGraphic, jug18d 0 /a "build.bmp"} To check files in but ignore changes

- 1 In the FileView pane of the Project Workspace window, select the files that you want to check in without incorporating changes.
- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Undo Check Out.

The Undo Check Out dialog box appears, with checks next to the files that you have selected. The list includes all checked-out files in the project directory, and you may check or uncheck any files in the list.

- 3 Choose the OK button.

The source-code control system changes the status of the files to checked in, but does not copy any of your changes to the master files. It does not record any differences. It also restores your local copy of the file so that it matches the master file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checking Files In and Merging Others' Changes

If your source-code control system does not support exclusive use, while you have had the file checked out, another author may have also checked the file out, made changes, and then checked the file in. In this case, before you check your local copy of the file in, you need to find out if there were changes by other authors. If so, you need to merge those changes into your local copy. You can then verify that all the changes are compatible and that none cause problems when you use the file. After you have verified the changes, you can check in the file.

{ewl msdncd, EWGraphic, jug19d 0 /a "build.bmp"} To check files in and merge others' changes

- 1 In the FileView pane of the Project Workspace window, select the files that you want to check in.
- 2 From the Tools menu, choose Source Control, and from the cascading menu, choose Get Latest Version.

The Get Latest Version dialog box appears, with checks next to the files that you have selected. The list includes all files in the project directory under source-code control, and you may check or uncheck any files in the list.

- 3 Choose the OK button.

The source-code control system copies master files to your local copies. If there is a file with changes in both your local copy and the master copy, your source-code control system notifies you that you have changes to merge. You then need to follow the recommended procedures in your source-code control system to reconcile the changes and verify those changes.

- 4 Repeat steps 1 through 3.

Remember that while you are verifying the last set of changes, another set may have appeared.

- 5 When your source-code control system reports that there are no more files to merge, select the files that you want to check in.
- 6 From the Tools menu, choose Source Control and from the cascading menu, choose Check In.

The Check In File(s) dialog box appears, with checks next to the files that you have selected. The list includes all files in the project directory, and you may check or uncheck any files in the list.

- 7 Choose the OK button.

You can also check in a file using the pop-up menu in the text editor windows. Press the right mouse button to display the menu, and choose Check In.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checking Files In When Closing the Workspace

You can choose to have Developer Studio prompt you to check in files when you close the current workspace.

{ewl msdncd, EWGraphic, jug20d 0 /a "build.bmp"} To check in files when closing the workspace

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Source Control tab.
- 3 Select the Check In Files When Closing The Workspace check box.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Maintaining Makefiles Under Source-Code Control

If you work in a group, you generally want to share the makefile for a project workspace with other members of your group. This ensures that everyone in the group can build the projects defined in the project workspace using the same files, settings, tools, and so on, as well as ensuring that everyone gets the changes to the makefile and builds with the most up-to-date settings.

Updating the makefile in a group setting requires some coordination among the members of the group in order for the process to work smoothly. In the optimal case, all members of the group get the makefile from source-code control when they open the project, but no one checks it out.

The following actions cause the makefile to change:

- Adding or deleting files.
- Adding or deleting projects, subprojects, or project configurations.
- Changing settings for any configuration.

If you want to take any of these actions, you need to take the following steps:

1. Plan the changes to make in your project workspace.
2. Check out the makefile.
3. Make the changes.
4. Save the changes to the makefile by choosing the Save All command from the File menu.
(Closing the project workspace or closing Developer Studio also saves the changes to the makefile.)
5. Check in the makefile.
6. Notify the members of the group that the makefile has changed.

At this point, the other members of the group need to close the project workspace, then reopen it and get the new version of the makefile. If they have set the option to prompt to get the latest versions of files, opening the project workspace reminds them to get the latest version.

If two or more members of the group simultaneously check out the makefile and revise it, checking the makefile in could require merging changes. Because Developer Studio may write multiple settings to a single line of the makefile, and changes to settings by different users may alter a single line in two (or more) different places, reconciling those changes manually could result in errors.

Note If your source-code control system supports exclusive check-outs, you should check the makefile out for exclusive use if you need to alter it.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Working with Resources

In Microsoft Developer Studio, a resource is an interface element that the user gains information from or manipulates to perform an action. Some basic resources are created for your project by Applet Wizard in Visual J++. For resource editing procedures common to all the editors, see Using the Resource Editors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Editors

The Developer Studio resource editors share techniques and interfaces to create and modify application resources quickly and easily. You can use the resource editors to create new resources, modify existing resources, copy existing resources, and delete old resources. The resource editors are functionally consistent for ease of use.

With Developer Studio you can edit all of the Microsoft Windows resources that your application uses:

- Image Files ([Using the Graphic Editor](#))
- Dialog boxes ([Using the Dialog Editor](#))
- Menus ([Using the Menu Editor](#))
- Template resources ([Using Resource Templates](#))

When you create or open a resource, the appropriate editor opens automatically; for example, graphical resources like [.GIF](#) and [.JPEG](#) image files. Dialog boxes are a combination of graphical components and text strings. Menus consist of text strings that appear in the menu bar.

The resource editors have many commands and procedures in common. For example, once you learn how to create and open a [dialog box](#), you know the steps for creating and opening any of the other resources. The most common resource editing activities are:

- [Viewing resources](#)
- [Creating new resources](#)
- [Using resource templates](#)
- [Editing resources](#)
- [Using the property pages](#)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Viewing Resources

There are different methods for viewing your project's resources. The type of resource you wish to view dictates the method you will use. Use one of the following to:

- View an image (.GIF or .JPEG) file
- View a dialog resource
- View a menu resource

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Viewing Image (.GIF or .JPEG) Files

**{ewl msdncd, EWGraphic, jug3r 0 /a "build.bmp"} To view an image resource
(.GIF or .JPEG file)**

- 1 Open the FileView pane by clicking on the FileView tab.
FileView appears (Figure 7.1 below)

- 2 Double-click on the image file you want to view.

Your file is displayed in the Image Editor Window of the Graphics Editor. To modify an image file, see Using the Graphics Editor.

Figure 7.1 The FileView Pane

{ewc msdncd, EWGraphic, jug3r 1 /a "uguideRESO.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Viewing Dialog Resources

{ewl msdncd, EWGraphic, jug4r 0 /a "build.bmp"} To view a Dialog resource

- 1** From the File menu, click Open.
The Open dialog box appears.
- 2** Navigate to the subdirectory where your resource template (.RCT) file is stored.
- 3** In the Files of Type drop-down list box, click Resource Files.
- 4** Double-click on the template (.RCT) file that contains the resource(s) you wish to view.
The Resource Template window appears (Figure 7.2 below).
When the Resource Template window is first displayed, the resource folder or its category folders may be condensed. You can expand the template or any category folder by clicking its plus sign (+).
- 5** To display all the dialog resources in the template (.RCT) file, click the plus (+) sign to the left of the dialog resources folder.
The names of all the dialog resources appear.
- 6** Double-click on the name of the resource you want to view.
The resource appears within the appropriate Developer Studio resource editor. If you want to modify the resource, see Using the Dialog Editor.

Figure 7.2 The Resource Template Window

{ewc msdncd, EWGraphic, jug4r 1 /a "uguideTMPL.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Viewing Menu Resources

{ewl msdncd, EWGraphic, jug5r 0 /a "build.bmp"} To view a menu resource

- 1** From the File menu, click Open.
The Open dialog box appears.
- 2** Navigate to the subdirectory where your resource template (.RCT) file is stored.
- 3** In the Files of Type drop-down list box, click Resource Files.
- 4** Double-click on the template (.RCT) file that contains the resource(s) you wish to view.
The Resource Template window appears (Figure 7.2).
When the Resource Template window is first displayed, the resource folder or its category folders may be condensed. You can expand the template or any category folder by clicking its plus sign (+).
- 5** To display all the menu resources in the template (.RCT) file, click the plus (+) sign to the left of the menu resources folder.
The names of all the menu resources appear.
- 6** Double-click on the name of the resource you want to view.
The resource appears within the appropriate Developer Studio resource editor. If you want to modify the resource, see Using the Menu Editor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a New Resource

There are several ways to create resources using Developer Studio. This section discusses:

- [Creating resources from the Insert Menu](#)
- [Creating resources from the Resource Template window](#)
- [Creating resources from the Resource Toolbar](#)

When you create a resource, Developer Studio assigns it a unique symbol name and value. If you need to change the symbol value, you can use the ID box on the resource's property page. For more information on the property page, use the Help button.

For more information on creating specific resource types, see:

- [Creating an Image File -- .GIF or .JPEG files](#)
- [Converting a Bitmap to an Image File](#)
- [Adding a menu resource](#)
- [Adding a dialog resource](#)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating Resources from the Insert Menu

{ewl msdncd, EWGraphic, jug7r 0 /a "build.bmp"} To create a new resource from the Insert menu

- 1** From the File menu, click Open.
The Open dialog box appears.
- 2** Navigate to the directory where your template (.RCT) file is stored.
- 3** Open your template (.RCT) file.
The Resource Template window appears (Figure 7.2).
- 4** From the Insert menu, click Resource.
The Insert Resource dialog box appears.
- 5** Select a resource from the Resource Type list box and click the OK button.
The appropriate resource editor window appears.
- 6** Create your new menu, dialog, or image resource.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating Resources from the Resource Template Window

{ewl msdncd, EWGraphic, jug8r 0 /a "build.bmp"} To create a new resource from the Resource Template window

- 1 From the File menu, click Open.
The Open dialog box appears.
- 2 Navigate to the directory where your template (.RCT) file is stored.
- 3 Open your template (.RCT) file.
The Resource Template window appears (Figure 7.2).
- 4 With the cursor in the Resource Template window, select Insert from the right mouse menu.
The Insert Resource dialog appears.
- 5 Select a resource from the Resource Type list box and choose the OK button.
The appropriate resource editor window appears.
- 6 Create your new menu, dialog, or image resource.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating Resources from the Resource Toolbar

{ewl msdncd, EWGraphic, jug9r 0 /a "build.bmp"} To create a new resource from the Resource Toolbar (Figure 7.3 below)

- 1 From the File menu, click Open.
The Open dialog box appears.
- 2 Navigate to the directory where your template (.RCT) file is stored.
- 3 Open your template (.RCT) file.
The Resource Template window appears (Figure 7.2).
- 4 From the View menu, click Toolbars.
The Toolbars dialog box appears.
- 5 Click the Resource check box.
The Resource toolbar appears.
- 6 Click the button for the type of resource you want to create.
The appropriate resource editor window appears.
- 7 Create your new menu, dialog, or image resource.

Figure 7.3 The Resource Toolbar

{ewc msdncd, EWGraphic, jug9r 1 /a "uguideTOOL.BMP"}

For information on displaying toolbars and customizing your workspace, see Showing and Hiding Toolbars.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Resource Templates

A template file is a copy of an edited resource that you can use to create additional resources. Resource templates can save time in developing additional resources or groups of resources that share a particular feature.

For instance, you might want to include a Help button and an icon of a company logo in several dialog boxes. You create a new template, and customize that template dialog box with the logo and Help button. Now, when you want to create a new dialog box, you can choose this template dialog box with the features already added.

{ewl msdncd, EWGraphic, jug10r 0 /a "build.bmp"} To create templates for resources

- 1 From the Insert menu, click Resource.
The Insert Resource dialog box (Figure 7.4 below) appears.
- 2 Select a resource from the Resource Type list box, and choose the OK button.
—or—
Copy a resource from another resource file. Hold down CTRL and drag the new resource to the resource template directory.

Figure 7.4 The Insert Resource Dialog Box

{ewc msdncd, EWGraphic, jug10r 1 /a "uguideRES.D.BMP"}

- 3 Modify the resource.
This resource, once saved as a template, can be copied numerous times to save effort on positioning controls, inserting text, and so on.
- 4 From the File menu, click Save As.
- 5 In the Save File As Type drop-down list box, select Resource Template (*.rct).
- 6 Select the TEMPLATE subdirectory under your MSDEV installation.
- 7 Click the OK button to save the template.
Repeat for any remaining templates.

{ewl msdncd, EWGraphic, jug10r 2 /a "build.bmp"} To create new resources from the templates

- 1 From the Insert menu, click Resource.
The Insert Resource dialog box appears.
- 2 Select a resource type. Click the resource icon to create a default resource template object.
—or—
Click the plus sign (+) next to a resource to move down the hierarchy to the template files grouped under that resource. Then click the specific template file under that resource to create a resource template object.
- 3 Click the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Editing Resources

The editors for the different resources share many of the same procedures. For more detailed information on editing the individual resources, see the topic for that resource.

{ewl msdncd, EWGraphic, jug11r 0 /a "build.bmp"} To open an existing image (.GIF or .JPEG) file for editing

- In FileView, select the image file you want to edit, and press ENTER.
—or—
Double-click the image file.
The resource editor window opens for editing.

For more information, see [Viewing Image Files](#).

{ewl msdncd, EWGraphic, jug11r 1 /a "build.bmp"} To open an existing menu or dialog resource for editing

- In the Resource Template window ([Figure 7.2](#)), expand the template hierarchy and select the menu or dialog resource you want to edit, and press ENTER.
—or—
Double-click the menu or dialog resource.
The menu editor window opens for editing menus; the dialog editor window opens for dialog boxes.

For more information see, [Viewing Menu Resources](#) and [Viewing Dialog Resources](#).

{ewl msdncd, EWGraphic, jug11r 2 /a "build.bmp"} To save an edited resource file

- From the File menu, click Save.
The resource is saved using its current name.
—or—
 - 1 From the File menu, click Save As.
 - 2 In the Drives list box, select the target drive.
 - 3 In the Directories list box, select the directory path.
 - 4 In the File Name box, type the name for the file.
 - 5 Click the OK button.The resource is saved using the Save As name.

{ewl msdncd, EWGraphic, jug11r 3 /a "build.bmp"} To delete an existing resource

- 1 In the Resource Template window, select the resource you want to delete.
- 2 From the Edit menu, click Delete.
The resource is deleted.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Property Pages

Property pages control the appearance and the behavior of resources and differ according to their purpose. For example, a bitmap resource property page contains information on the ID, and filename, as well as a preview of the resource. But a property page for a button control in a dialog box contains several tabs of information, General and Extended Styles, each with many style bits to modify the control's behavior.

Note Whenever you make a change on a property page, it is made immediately. You cannot cancel any changes made on a property page.

Manipulating a Property Page

You can use any of the editing keyboard shortcut keys to cut, copy, and paste text. In general these shortcut keys can be used in any edit control on the property page.

You can control the behavior of the Properties window to suit your working style or the nature of the resource editing task. Use the Pushpin button in the upper-left corner of the property page to control the page.

But ton pos itio n	Result
{ew c ms dnc d, EW Gr ap hic, jug 12r 0 / a "ug uid e.B MP "} }	When the button is in the down position, the Properties window stays visible even when you are working in another window. This is convenient if, during an editing session, you want to move back and forth frequently between setting properties and editing objects. Pressing ENTER after you change a value in the Properties window

returns you to
the editing
window but
leaves the
Properties
window
visible.

{ew When the
c button is in the
ms up position,
dnc you can
d, dismiss the
EW active
Gr Properties
ap window by
hic, pressing
jug ENTER or ESC.
12r This is useful
1 / if you want to
a concentrate
"ug on working in
uid an editing
e.B window but
MP need to bring
"} up the
Properties
window briefly
to change one
or two values.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Dialog Editor

The Microsoft Developer Studio dialog editor helps with the creation or editing of a dialog box resource. You can place, arrange, or activate controls; and test the dialog box. Dialog boxes can be stored as templates.

With the dialog editor, you can:

- Add, arrange, or edit controls.
- Change the tab order or accelerator keys.
- Use guides in the dialog layout.
- Test a dialog box.

You open the Dialog Editor by adding a new dialog resource or by opening an existing dialog resource.

You can use resource templates to create dialog boxes to use later or copy dialog box resources. For more information, see Using Resource Templates.

Figure 8.1 The Dialog Editor

```
{ewc msdncd, EWGraphic, jug0y 0 /a "uguideDIAL.BMP"}
```

Tip While using the dialog editor, in many instances you can click the right mouse button to display a pop-up menu of frequently used commands. The commands available depend on what the pointer is pointing to. For example, if you click while pointing to a dialog box, the pop-up menu shows the Check Mnemonics and Properties commands.

For information about common resource edit procedures such as creating new resources, opening existing resources, and deleting resources, see Working with Resources.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding and Editing Controls in a Dialog Box

One of the first steps to creating a new dialog box (or making a dialog box template) is to add controls to the dialog box. Controls can be edited to fit a certain size, shape, or alignment, or they can be moved around to work within the dialog box.

This section focuses on:

- Types of controls in dialog boxes
- Adding controls to dialog boxes
- Selecting specific controls or groups of controls
- Sizing individual controls

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Types of Controls

With the dialog editor you can create dialog boxes that include the standard control types shown on the Controls toolbar in Figure 8.2.

Figure 8.2 The Controls Toolbar

```
{ewc msdncd, EWGraphic, jug2y 0 /a "uguideCONT.BMP"}
```

By default the Controls toolbar is displayed when the dialog editor is open, but you can modify this behavior.

{ewl msdncd, EWGraphic, jug2y 1 /a "build.bmp"} To hide the Controls toolbar

- Click the close box in the upper-left corner of the Controls toolbar.

{ewl msdncd, EWGraphic, jug2y 2 /a "build.bmp"} To show the Controls toolbar

- 1 From the View menu, choose Toolbars.
- 2 Select the Controls check box, and then choose Close.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding Controls

You add controls to a dialog box by using the Controls toolbar to choose the control you want and drag the control to the dialog box. When displayed, the toolbar stays positioned above other open windows in your workspace. Controls that are currently supported for Visual J++ dialog boxes are the Static Text, Edit Box, Group Box, Button, Check Box, Radio Button, Combo Box, List Box, Horizontal scroll bar and Vertical scroll bar.

Note The Dialog Editor allows you to place all of the controls on the Controls toolbar onto your dialog box template; Resource Wizard can generate .CLASS files for only those controls listed in Table 10.1.

The fastest way to add controls to a dialog box, reposition existing controls, or move controls from one dialog box to another is to use the drag-and-drop method. (See Figure 8.3.) The control's position is outlined in a dotted line until it is dropped into the dialog box. When you add a control to a dialog box with drag-and-drop, the control is given a standard height appropriate to that type of control.

Figure 8.3 Dragging a Control from the Controls Toolbar

```
{ewc msdncd, EWGraphic, jug3y 0 /a "uguideDRAG.BMP"}
```

You can also add a new control by clicking the Controls toolbar button for the control you want and:

- “Drawing” the control in the dialog box. This is a good method when you want to specify the initial size of the object. Just place the pointer where you want the upper-left corner of the control to be. Drag the pointer to the right and downward to the appropriate size for that control.
- Clicking the dialog box at the location you want. This is an alternative method to dragging and dropping.

Holding down CTRL when selecting a control from the Controls toolbar places multiple controls using either method listed above. Pressing ESC stops placing controls.

When you add a control to a dialog box or reposition it, its final placement may be determined by guides or margins, or whether you have the Grid turned on. For more information about guides and margins, see Using Guides and Margins. For information about the Grid and other placement and alignment tools, see Arranging Controls.

When you have added a control to the dialog box, you can change its caption or any of its other properties in its property page.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Selecting Controls

To move, copy, delete, or align controls, you select them and then perform the operation you want. In most cases, you need to select more than one control to use the sizing and alignment tools on the Dialog toolbar.

When a control is selected, it has a shaded border around it with solid (active) or hollow (inactive) “sizing handles,” small squares that appear in the selection border.

When you are sizing or aligning multiple controls, the dialog editor uses the “dominant control” to determine how the other controls are sized or aligned. When multiple controls are selected, the dominant control has solid sizing handles; all the other selected controls have hollow sizing handles.

{ewl msdncd, EWGraphic, jug4y 0 /a "build.bmp"} To select multiple controls

- 1 From the Controls toolbar, select the pointer tool.
- 2 Drag to draw a selection box around the controls you want to select (Figure 8.4). Controls partially outside the selection box are not selected.
When you release the mouse button, all controls inside the selection box are selected.

Figure 8.4 Selecting Multiple Controls

{ewc msdncd, EWGraphic, jug4y 1 /a "uguideSELE.BMP"}

Once you have selected one or more controls, you can remove or add individual controls without disturbing the selection as a whole.

- 1 Hold down the SHIFT key and click the control you want to remove from or add to the existing selection.

{ewl msdncd, EWGraphic, jug4y 2 /a "build.bmp"} To change the dominant control when more than one control is selected

- Hold down the CTRL key and click the control you want to use to influence the size or location of the others.
The sizing handles change from hollow to solid. All further resizing or alignment is based on this control.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Sizing Individual Controls

Use the sizing handles to resize a control. When the pointer is positioned on a sizing handle, it changes shape to indicate the direction in which the control will be resized (see Figure 8.5). Active sizing handles are solid; if a sizing handle is hollow, the control cannot be resized along that axis.

Figure 8.5 Sizing a Control

```
{ewc msdncd, EWGraphic, jug5y 0 /a "uguideSIZI.BMP"}
```

You can also change the size of a control by snapping the control to guides or margins, or by moving a snapped control and guide away from another. For more information, see Using Guides and Margins. The final shape of the control may be affected by whether or not you have the Grid turned on. For more information, see Using the Layout Grid.

{ewl msdncd, EWGraphic, jug5y 1 /a "build.bmp"} To size a control

- 1 Click the control, or select it with the TAB key.
- 2 Drag the sizing handles to change the size of the control:
 - Sizing handles at the top and sides change the horizontal or vertical size.
 - Sizing handles at the corners change both horizontal and vertical size.

—or—

Hold down the SHIFT key and use the ARROW keys to resize the control one dialog unit (DLU) at a time.

As you type a caption to text within a control, the control will resize to fit the text caption. This function can be disabled by manually resizing the control with the sizing handles. To return to the automatic resizing of a control to fit the text within it, choose Size To Content from the Layout menu.

When you select a drop-down combo box or drop-down list box to size it, only the right and left sizing handles are active (see Figure 8.6). Use these handles to set the width of the box as it is initially displayed.

You can also set the vertical size of the drop-down portion of the box.

Figure 8.6 Sizing the Drop-down Portion of a Combo Box

```
{ewc msdncd, EWGraphic, jug5y 2 /a "uguideSTDP.BMP"}
```

{ewl msdncd, EWGraphic, jug5y 3 /a "build.bmp"} To set the size of the combo box drop-down area

- 1 Click the drop-down arrow at the right of the combo box (Figure 8.6).
The outline of the control changes to show the size of the combo box with the drop-down area extended.
- 2 Use the bottom sizing handle to change the initial size of the drop-down area.
- 3 Click the drop-down arrow again to close the drop-down portion of the combo box.

You can resize a group of controls based on the size of the dominant control. You can also resize a control based on the dimensions of its caption text.

{ewl msdncd, EWGraphic, jug5y 4 /a "build.bmp"} To make controls the same

width, height, or size

1 Select the controls you want to resize.

2 Make sure the correct dominant control is selected.

The final size of the controls in the group depends on the size of the dominant control. For more information on selecting the dominant control, see [Selecting Controls](#).

3 Choose one of the following tools on the Dialog toolbar:

- Make Same Width
- Make Same Height
- Make Same Size

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Formatting the Layout of a Dialog Box

The dialog editor contains special tools for layout to help in arranging controls in the correct place and alignment. Some of these tools are contained on the Dialog toolbar, like guides and the Grid.

You can use the dialog editor in three different states for moving controls: with the guides and margins on (default setting), with Grid on, or plain, with no snapping or alignment features on at all.

You can:

- Arrange the controls using the Dialog toolbar.
- Align controls with each other or by spacing.
- Use guides and margins to align controls inside the dialog box.
- Use Grid to place controls inside the dialog box.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Arranging Controls

The dialog editor provides layout tools that align and size controls automatically. For most tasks, you can use the Dialog toolbar (see Figure 8.7). All commands are also available on the Layout menu, and most have shortcut keys.

Figure 8.7 Dialog Toolbar

{ewc msdncd, EWGraphic, jug7y 0 /a "uguideDToo.BMP"}

Many layout commands are available only when more than one control is selected. For information on selecting more than one control, see Selecting Controls.

The location, height, and width of the current control is displayed in the lower-right corner of the Developer Studio status bar (see Figure 8.8). When more than one control is selected, the position indicators show the position of the dominant control (the control with solid sizing handles). When the dialog box is selected, the status bar displays the position of the dialog box and its height and width.

Figure 8.8 Dialog Editor Position Indicators

{ewc msdncd, EWGraphic, jug7y 1 /a "uguideDEDI.BMP"}

The location and size of a dialog box, as well as the location and size of controls within it, are measured in dialog units (DLUs). A DLU is based on the size of the dialog box font, normally 8-point MS Sans Serif. A horizontal DLU is the average width of the dialog box font divided by four. A vertical DLU is the average height of the font divided by eight.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Aligning Controls

Once controls are in place, the dialog editor offers a variety of ways to refine their positions. You can:

- Align a group of controls along their left, right, top, or bottom edges.
- Align a group of controls on their center, either horizontally or vertically.
- Even the spacing between a group of three or more controls.
- Center one or more controls in the dialog box, vertically or horizontally.
- Automatically give command buttons a standard position along the bottom or on the right of the dialog box.

{ewl msdncd, EWGraphic, jug8y 0 /a "build.bmp"} To align controls

- 1 Select the controls you want to align.
- 2 Make sure the correct dominant control is selected.
The final position of the group of controls depends on the position of the dominant control. For more information on selecting the dominant control, see Selecting Controls.
- 3 From the Layout menu, choose Align Controls, and then choose one of the following alignments:
 - The Left command: Aligns the selected controls along their left side.
 - The Right command: Aligns the selected controls along their right side.
 - The Top command: Aligns the selected controls along their top edges.
 - The Bottom command: Aligns the selected controls along their bottom edges.

{ewl msdncd, EWGraphic, jug8y 1 /a "build.bmp"} To align controls on their center, vertically or horizontally

- 1 Select the controls you want to center.
- 2 Make sure the correct dominant control is selected.
The final position of the group of controls depends on the position of the dominant control. For more information on selecting the dominant control, see Selecting Controls.
- 3 From the Layout menu, choose Align Controls, and then choose Vert. Center or Horiz. Center.

{ewl msdncd, EWGraphic, jug8y 2 /a "build.bmp"} To even the spacing between controls

- 1 Select the controls you want to rearrange.
- 2 From the Layout menu, choose Space Evenly, and then choose one of the following spacing alignments:
 - Across: Controls are spaced evenly between the leftmost and the rightmost control selected.
 - Down: Controls are spaced evenly between the topmost and the bottom most control selected.

{ewl msdncd, EWGraphic, jug8y 3 /a "build.bmp"} To center controls in the dialog box

- 1 Select the control or controls you want to rearrange.
- 2 From the Layout menu, choose Center In Dialog, and then choose one of the following arrangements:
 - Vertical: Controls are centered vertically in the dialog box.

- Horizontal: Controls are centered horizontally in the dialog box.

{ewl msdncd, EWGraphic, jug8y 4 /a "build.bmp"} To arrange command buttons along the right or bottom of the dialog box

- 1 Select one or more command buttons.
- 2 From the Layout menu, choose Arrange Buttons, and then choose one of the following arrangements:
 - Right
 - Bottom

The selected buttons are positioned in a standard arrangement along the bottom or right side of the dialog box. If a control other than a command button is selected, its position is not affected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Guides and Margins

Whether you are moving controls, adding controls, or rearranging a current layout, guides can help you align controls accurately within a dialog box. Guides appear as blue dotted lines across the dialog box displayed in the editor and corresponding arrows in the rulers.

When you create a dialog box, four margins are provided. Margins are modified guides, appearing as blue dotted lines.

You can:

- Align controls on a guide or move controls with a guide.
- Disable the guides or move the guides without the controls.

Figure 8.9 shows the dialog editor with guides and margins.

Figure 8.9 Dialog Editor with Guides and Margins

```
{ewc msdncd, EWGraphic, jug9y 0 /a "uguideGIDS.BMP"}
```

{ewl msdncd, EWGraphic, jug9y 1 /a "build.bmp"} To create and set a guide

- 1 Click anywhere within the rulers to create a guide.
- 2 Drag the guide into position.

The number of DLUs is displayed in the ruler and below on the Developer Studio status bar. After the guide is dropped into position, hold the cursor over the guide's arrow in the ruler to see the exact position of the guide.

{ewl msdncd, EWGraphic, jug9y 2 /a "build.bmp"} To delete a guide

- Drag the guide out of the dialog box that is being edited.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Aligning Controls on a Guide

The sizing handles of controls snap to guides when the controls are moved, and guides snap to controls (if there are no controls previously snapped to the guide). When a guide is moved, controls that are snapped to it move as well. Controls snapped to more than one guide are resized when one of the guides is moved.

The tick marks in the rulers that determine the spacing of guides and controls are determined by dialog units (DLUs). A DLU is based on the size of the dialog box font, normally 8-point MS Sans Serif. A horizontal DLU is the average width of the dialog box font divided by four. A vertical DLU is the average height of the font divided by eight.

{ewl msdncd, EWGraphic, jug10y 0 /a "build.bmp"} To move guides

- Drag the guide to the new position.
The coordinates of the guide are displayed in the status bar at the bottom of the Developer Studio window and in the ruler. Move the pointer over the arrow in the ruler to display the exact position of the guide.

{ewl msdncd, EWGraphic, jug10y 1 /a "build.bmp"} To move margins

- Drag the margin to the new position.
—or—
Move the gray spacing block in the ruler adjoining the margin.
To make a margin disappear, move the margin to a zero position. To bring that margin back, place the pointer over the margin's zero position and move the margin into position.

{ewl msdncd, EWGraphic, jug10y 2 /a "build.bmp"} To size a group of controls with guides

- 1 Snap one side of the control (or controls) to a guide.
- 2 Drag a guide to the other side of the control (or controls).
If necessary with multiple controls, size each to snap to the second guide.
- 3 Move either guide to size the control (or controls) on that side.

{ewl msdncd, EWGraphic, jug10y 3 /a "build.bmp"} To change the intervals of the tick marks

- 1 From the Layout menu, choose Guide Settings.
The Guide Settings dialog box appears.
- 2 In the Grid Spacing box, specify the new width and height in DLUs.
- 3 Choose OK.

Disabling the Guides

You can use special keys in conjunction with the mouse to disable the snapping effect of the guides. Using the ALT key disables the snapping effects of the guide selected. Moving a guide with the SHIFT key prevents snapped controls from moving with the guide.

{ewl msdncd, EWGraphic, jug10y 4 /a "build.bmp"} To disable the snapping effect of the guides

- Drag the control while holding down the ALT key.

{ewl msdncd, EWGraphic, jug10y 5 /a "build.bmp"} To move guides without moving the snapped controls

- Drag the guide while holding down the SHIFT key.

{ewl msdncd, EWGraphic, jug10y 6 /a "build.bmp"} To clear all the guides

- 1 Click the right mouse button in the ruler bar.
- 2 From the pop-up menu, choose Clear All.

{ewl msdncd, EWGraphic, jug10y 7 /a "build.bmp"} To turn off the guides

- 1 From the Layout menu, choose Guide Settings.
The Guide Settings dialog box appears.
- 2 Under Layout Guides, select None.
- 3 Choose the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Layout Grid

When you are placing or arranging controls in a dialog box, you can use the layout grid for more precise positioning. When the grid is turned on, controls appear to “snap to” the dotted lines of the grid as if magnetized. You can turn this “snap to grid” feature on and off and change the size of the layout grid cells.

{ewl msdncd, EWGraphic, jug12y 0 /a "build.bmp"} To turn the Grid on or off

- 1 From the Layout menu, choose Guide Settings.
- 2 Select or clear the Grid radio button.

You can still control Grid in individual dialog editor windows using the Toggle Grid button on the Dialog toolbar.

{ewl msdncd, EWGraphic, jug12y 1 /a "build.bmp"} To change the size of the layout grid

- 1 From the Layout menu, choose Guide Settings.
- 2 Type the height and width in DLUs for the cells in the grid. The minimum height or width is 4 DLUs. For more information on DLUs, see Arranging Controls.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Editing the Dialog Box

Each dialog box has a property page, a tab order, and mnemonic keys. The tab order is the order that the focus moves from when using the TAB key. Alternatively, a keyboard user can press a mnemonic key to move the input focus from one control to another.

You can:

- Change the tab order for the input focus.
- Define the mnemonic keys for the input focus.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Changing the Tab Order

The tab order is the order in which the TAB key moves the input focus from one control to the next within a dialog box. Usually the tab order proceeds from left to right in a dialog box, and from top to bottom. Each control has a property page with a Tabstop check box used to determine whether a control actually receives input focus or not.

Even controls that do not have the Tabstop property set need to be part of the tab order. This can be important, for example, when you define mnemonics for controls that do not have captions. Static text that contains a mnemonic for a related control must immediately precede the related control in the tab order.

Note If your dialog box contains overlapping controls, changing the tab order may change the way the controls are displayed. Controls that come first in the tab order are always displayed on top of any overlapping controls that follow them in the tab order.

{ewl msdncd, EWGraphic, jug14y 0 /a "build.bmp"} To change the tab order for all controls in a dialog box

- 1 From the Layout menu, choose Tab Order.
A number in the upper-left corner of each control shows its place in the current tab order.
- 2 Set the tab order by clicking each control in the order you want the TAB key to follow.
- 3 Press ENTER to exit Tab Order mode.

{ewl msdncd, EWGraphic, jug14y 1 /a "build.bmp"} To change the existing tab order

To change the existing tab order, specify the starting control; that is, select the control *prior to* the one where you want the changed order to begin. The selected control determines the number of the control you click next. For example, if you are in Tab Order mode, and control number 3 is selected, the next control you click is set to number 4.

- 1 From the Layout menu, choose Tab Order.
- 2 Specify where the change in order will begin. To do this, hold down the CTRL key and click the control *prior to* the one where you want the changed order to begin.
For example, if you want to change the order of controls 7 through 9, select control 6 first.

Note To set a specific control to number 1 (first in the tab order), double-click the control.

- 3 Reset the tab order by clicking the controls in the order you want the TAB key to follow.
- 4 Press ENTER to exit Tab Order mode.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Defining Mnemonic Keys

Normally, keyboard users move the input focus from one control to another in a dialog box with the TAB and ARROW keys. However, you can define a mnemonic key that allows users to choose a control by pressing a single key.

Note All the mnemonics within a dialog box should be unique.

{ewl msdncd, EWGraphic, jug15y 0 /a "build.bmp"} To define a mnemonic key for a control with a visible caption (command buttons, check boxes, and radio buttons)

- 1 Select the control.
- 2 To open the control's property page:
 - from the Edit menu, choose Properties.

—or—

 - Press ALT+ENTER.
- 3 In the Caption box, type an ampersand (&) in front of the letter you want as the mnemonic for that control.

An underline appears in the displayed caption to indicate the mnemonic key.

{ewl msdncd, EWGraphic, jug15y 1 /a "build.bmp"} To define a mnemonic for a control without a visible caption

- 1 Make a caption for the control by using a static text control. In the static text caption, type an ampersand (&) in front of the letter you want as the mnemonic.
- 2 Make sure the static text control immediately precedes the control it labels in the tab order.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Testing a Dialog Box

You can simulate the run-time behavior of a dialog box from within the dialog editor without compiling your program. This gives you immediate feedback on how the layout of controls appears and performs and thus speeds up the user-interface design process.

When you are in test mode, you can:

- Type text, select from combo-box lists, turn options on and off, and choose commands.
- Test the tab order.
- Test the grouping of controls, such as radio buttons or check boxes.
- Test the dialog box's keyboard shortcuts (for controls that have mnemonic keys defined for them).

When you test a dialog box, it is usually displayed at a location relative to the main Developer Studio program window. If the dialog box's Absolute Align property is selected, the dialog box is displayed at a position relative to the upper-left corner of the screen.

{ewl msdncd, EWGraphic, jug16y 0 /a "build.bmp"} To test a dialog box

- 1 From the Layout menu, choose Test.
- 2 To end the test session, do one of the following actions:
 - Press ESC.
 - Close the dialog box using its Control-menu box.
 - Choose a command button with a symbol name of IDOK or IDCANCEL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Menu Editor

Menus allow you to arrange commands in a logical, easy-to-find fashion. With the Microsoft Developer Studio menu editor, you can create and edit menus by working directly with a menu bar that closely resembles the one in your finished application.

With the menu editor, you can:

- Create standard menus and commands.
- Assign shortcut keys.
- Move menus or commands from one place to another.

You open the Menu Editor by adding a new menu resource or by opening an existing menu resource.

Tip While using the menu editor, in many instances you can click the right mouse button to display a pop-up menu of frequently used commands. The commands available depend on what the pointer is pointing to. For example, if you click while pointing to a menu item, the pop-up menu shows Cut, Copy, and Paste commands as well as the property page for the selected item.

For information about common resource edit procedures such as creating new resources, opening existing resources, and deleting resources, see Working with Resources.

The menu editor, with its various components labeled, is shown in Figure 9.1

Figure 9.1 Menu Editor Components

{ewc msdncd, EWGraphic, jug0n 0 /a "uguideTERM.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating Menus or Menu Items

You can create menus, cascading menus, and menu commands on the menu bar in the menu editor.

{ewl msdncd, EWGraphic, jug1n 0 /a "build.bmp"} To create a menu on the menu bar

- 1 Select the new-item box (an empty rectangle) on the menu bar (see Figure 9.2). Or move the new-item box to a blank spot with the RIGHT ARROW and LEFT ARROW keys.

Figure 9.2 Menu Editor New-Item Boxes

{ewc msdncd, EWGraphic, jug1n 1 /a "uguideNEWI.BMP"}

- 2 Type the name of the menu.

When you start typing, FOCUS automatically shifts to the Menu Item property page, and the text you type appears both in the Caption box and in the menu editor window.

You can define a mnemonic key that allows the user to select the menu with the keyboard.

Type an ampersand (&) in front of a letter to specify it as the mnemonic. Make sure all the mnemonics associated with a specific drop-down menu bar are unique.

Once you have given the menu a name on the menu bar, the new-item box shifts to the right, and another new-item box opens below for adding menu items.

Note To create a single-item menu on the menu bar, clear the Pop-up check box on the Menu Item property page.

{ewl msdncd, EWGraphic, jug1n 2 /a "build.bmp"} To create a menu item

- 1 First, create a menu according to the steps outlined in the previous procedure.
- 2 Select the menu's new-item box.
—or—
Select an existing menu item and press the INS KEY. The new-item box is inserted before the selected item.
- 3 Type the name of the menu item. When you start typing, FOCUS automatically shifts to the Menu Item property page, and the text you type appears in the Caption box.

You can define a mnemonic key that allows the user to select the menu command. Type an ampersand (&) in front a letter to specify it as the mnemonic. The user can select the menu command by typing that letter.

- 4 In the ID box, type the menu item ID, or select an existing command identifier. If you don't specify an ID, Developer Studio will generate an ID for you based on the command name.
- 5 On the property page, select the menu item styles that apply.
- 6 Press ENTER to complete the menu item.

The new-item box is selected so you can create additional menu items.

{ewl msdncd, EWGraphic, jug1n 3 /a "build.bmp"} To create a cascading (hierarchical) menu

- 1 Select the new-item box on the menu where you want the cascading menu to appear. Then type the name of the menu item that, when selected, will cause the cascading menu to appear.

When you start typing, FOCUS automatically shifts to the Menu Item property page, and the text you type appears in the Caption box.

–or–

Select an existing menu item that you want to be the parent item of the cascading menu, and double-click.

- 2** On the property page, select the Pop-up check box. This marks the menu item with the cascading menu symbol, and a new-item box appears to its right.
- 3** Add additional menu items to the cascading menu according to the instructions in the previous procedure.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Selecting Menus and Menu Items

{ewl msdncd, EWGraphic, jug2n 0 /a "build.bmp"} To select a menu and display its menu items

- Click the menu caption on the menu bar or the parent item of the cascading menu. Then click the menu item you want.
—or—
Move to the menu caption with the TAB (move right) and SHIFT+TAB (move left) keys or the RIGHT ARROW and LEFT ARROW keys.

{ewl msdncd, EWGraphic, jug2n 1 /a "build.bmp"} To select one or more menu items

- 1 Click the menu or cascading menu you want.
Its menu items are displayed.
- 2 Click to select a single menu item, or press the SHIFT key while clicking to select multiple menu items. Hold down the SHIFT key and click a selected menu item to deselect it.
—or—
With the pointer outside the menu, drag to draw a selection box around the menu items you want to select.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Moving and Copying Menus and Menu Items

{ewl msdncd, EWGraphic, jug3n 0 /a "build.bmp"} To move or copy menus or menu items using drag-and-drop

- 1 Drag or copy the item you want to move to:
 - A new location on the current menu.
 - A different menu. (You can navigate to other menus by dragging the mouse pointer over them.)
- 2 Drop the menu item when the insertion guide shows the position you want. Figure 9.3 illustrates this process.

Figure 9.3 Moving a Menu to a Cascading Menu

{ewc msdncd, EWGraphic, jug3n 1 /a "uguideCASC.BMP"}

{ewl msdncd, EWGraphic, jug3n 2 /a "build.bmp"} To move or copy menus or menu items using the menu commands

- 1 Select one or more menus or menu items.
- 2 From the Edit menu, choose Cut (to move) or Copy.
- 3 If you are moving the items to another menu resource or resource script file, make that menu editor window active.
- 4 Select the position of the menu or menu item you want to move or copy to.
- 5 From the Edit menu, choose Paste. The moved or copied item is placed before the item you select.

Note You can also drag, copy, and paste to other menus in other menu windows.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Resource Wizard

With traditional Windows development tools, user-interface elements like menus and dialogs are specified using resource files that are separate from the source files. In Java, by contrast, menus and dialogs are created in the source code, using Java's Abstract Window Toolkit (AWT). Resource Wizard is a tool for converting Windows resource template (.RCT) files and Windows resource (.RES) files into equivalent Java code.

Before proceeding, you should have previously created a new applet with Applet Wizard.

The following topics discuss the specifics of using the Visual J++ Resource Wizard:

- Creating a resource template
- Adding Dialog Resources
- Adding Menu Resources
- Using Java GUI Components
- Running Resource Wizard
- What Resource Wizard Generates
- Using the Resource Wizard-generated dialog
- Using the Resource Wizard-generated menu
- Seeing the results

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating a Resource Template

The Java Resource Wizard generates Java GUI component code for specific Windows controls in template (.RCT) and resource (.RES) files. These Java components are defined in the Java's Abstract Window Toolkit (AWT). If a comparable Java class does not exist for a Windows control, no code will be generated for the Window control. For a list of controls that are supported, see Using Java GUI Components. To create a resource template that Resource Wizard uses as input, use the Microsoft Developer Studio dialog and menu editors to create specific resources that are stored in template (.RCT) files.

{ewl msdncd, EWGraphic, jug1s 0 /a "build.bmp"} To create a resource template

- 1 From the File menu, choose New.

The New dialog box appears.

- 2 Choose Resource Template.

Visual J++ opens a resource template window with an empty template folder. This is the visual representation of your resource template (.RCT) file.

- 3 To save the file, choose Save from the File menu.

The Save As dialog appears.

- 4 Navigate to the subdirectory where you want to save the resource template.

- 5 Change the name of the file if you wish, but keep the .RCT extension.

- 6 Click the save button.

To add dialog and menu resources to the resource template file, see Adding Dialog Resources and Adding Menu Resources.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding Dialog Resources

To assist the user in using your applet or application, you will probably want to add Graphical User Interface (GUI) components to your project. Although you can add these features programmatically, you will find using Microsoft Developer Studio's resource editors and the Java Resource Wizard to be a simpler alternative. Before you can add either dialog or menu resources to your project with Resource Wizard, you will need a resource template (.RCT) file, see Creating a Resource Template. This section discusses adding dialog resources; the following section discusses adding menu resources.

{ewl msdncd, EWGraphic, jug2s 0 /a "build.bmp"} To add a dialog resource to a resource template (.RCT) file

- 1** Open the resource template (.RCT) file, by selecting Open from the File menu.
The Open dialog appears.
- 2** Navigate to the subdirectory where your resource template file is saved and open it.
The resource template folder appears in the Resource Template window.
- 3** From the Insert menu, choose Resource.
—or—
From the right mouse menu, choose Insert.
The Insert Resource dialog box appears.
- 4** Choose Dialog.
Visual J++ opens a dialog editing window.
- 5** Press ALT+ENTER to display the property page for the dialog. Change the ID and the caption of the dialog to an appropriate name and caption for your dialog box.
- 6** Add controls to the dialog using the Controls palette.
While the Developer Studio dialog editor allows you to add any type of control to the dialog, only controls that have analogs in Java's Abstract Window Toolkit (AWT) will be converted by the Resource Wizard. For information about which controls are supported, see Using Java GUI Components.
- 7** Close the dialog editing window by clicking the Close button of the top-level menu bar.
- 8** From the File menu, choose Save.
The file Save As dialog box appears.
- 9** Change the directory to the "C:\msdev\projects\NewApp\template", or any directory where you save your resource template files.
The default filename that Visual J++ suggests is "Templx.rct," where "x" is the next sequential number indicating how many new templates you have created. You can change this to a name that is appropriate for your resource template.
- 10** Choose OK.

For information on using Developer Studio's dialog editor, see Using the Dialog Editor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding Menu Resources

To assist the user in using your applet or application, you will probably want to add Graphical User Interface (GUI) components to your project. Although you can add these features programmatically, you will find using Microsoft Developer Studio's resource editors and the Java Resource Wizard to be a simpler alternative. Before you can add either dialog or menu resources to your project with Resource Wizard, you will need a resource template (.RCT) file, see Creating a Resource Template. This section discusses adding menu resources; the previous section discusses adding dialog resources.

{ewl msdncd, EWGraphic, jug3s 0 /a "build.bmp"} To add a menu resource to a resource template (.RCT) file

- 1 Open the resource template (.RCT) file, by selecting Open from the File menu.
The Open dialog appears.
- 2 Navigate to the subdirectory where your resource template file is saved and open it.
The resource template folder appears in the Resource Template window.
- 3 From the Insert menu, choose Resource.
—or—
From the right mouse menu, choose Insert.
The Insert Resource dialog box appears.
- 4 Choose Menu.
Visual J++ opens the menu editing window.
- 5 Press ALT+ENTER to display the Menu Properties page for the menu. Change the Caption of the top-top level menu item to an appropriate name for your menu resource.
- 6 Add new menus and menu items.
For information on using the Developer Studio menu editor, see Using the Menu Editor.
- 7 Close the menu editing window by clicking the Close button of the top-level menu bar.
- 8 From the File menu, choose Save.
The file Save As dialog box appears.
- 9 Change the directory to the "C:\msdev\projects\NewApp\template," or any directory where you save your resource template files.
The default filename that Visual J++ suggests is "Templx.rct;" where "x" is the next sequential number indicating how many new templates you have created. You can change this to a name that is appropriate for your resource template.
- 10 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Java GUI Components

Whether you are using a new resource template (.RCT) file or using an existing Windows resource (.RES) file, remember these files need to be converted to Java code to produce the equivalent Java Graphical User Interface (GUI) components. The Resource Wizard is a tool that can do this conversion for you. When using the Resource Wizard, you will find that not all controls in the Dialog Editor's toolbox have an equivalent component in Java's Abstract Window Toolkit (AWT) class library. Table 10.1 lists the Windows controls that are supported and their corresponding Java component classes.

Note Although the Dialog Editor allows you to put all the controls in the toolbox on your dialog template, only those shown in Table 10.1 will be converted to Java GUI components by the Resource Wizard.

Table 10.1 Java GUI Components Supported by Resource Wizard

Dialog Editor Toolbox Control	Windows Resource	Java Component (java. awt.*)
{ewc msd ncd, EW Graphic, jug4 s 0 /a "ugui deST AT.B MP"}	Static Text control	Label class
	Edit Box control	TextFiled class (or TextArea for multiline controls)

deE
DIT.
BMP
"}
"

Button **Butto**
control **n**
class

{ewc
msd
ncd,
EW
Grap
hic,
jug4
s
2 /a
"ugui
deB
UTT.
BMP
"}
"

Check **Check**
Box **box**
control class

{ewc
msd
ncd,
EW
Grap
hic,
jug4
s
3 /a
"ugui
deC
HKB
X.B
MP"}
"

Radio **Check**
Button **box**
control (Reso
urce
[Wiza](#)
[rd](#)
gener
ates
and
adds
the
control
to a
Check

{ewc
msd
ncd,
EW
Grap
hic,
jug4
s
4 /a
"ugui
deR

ADI.
BMP
"}

boxGroup
member

{ewc
msd
ncd,
EW
Graphic,
jug4
s
5 /a
"ugui
deLl
ST.B
MP"}

List
Box
control

List
class

{ewc
msd
ncd,
EW
Graphic,
jug4
s
6 /a
"ugui
deC
OMB
.BM
P"}

ComboBox
Choice
class

{ewc
msd
ncd,
EW
Graphic,
jug4
s
7 /a
"ugui
deH
ORI.
BMP

Horizontal
Scrollbar
bar
control

Scrollbar
class
(Resource
Wizard
sets
the
HORIZONTAL
property.)


```

"}
    Vertical Scroll Scrollbar
    {ewc al class
    msd Scroll (Resource
    ncd, bar urce
    EW control Wizard
    Graphic sets
    hic, the
    jug4 VERTICAL
    s CAL
    8 /a property.)
    "ugui
    deV
    ERT.
    BMP
    "}

```

For a few types of control, certain properties are maintained in the generated Java code. For static text controls, you can specify right, left, or center alignment of text. For edit box controls, you can specify the password style for the control. For list box controls, you can specify single or multiple selection.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Running Resource Wizard

The Resource Wizard reads a Windows template (.RCT) and resource (.RES) files and generates Java code to produce a user interface that uses the Java Graphical User Interface (GUI) component classes. For a list of the controls that can be converted from a Windows template or resource file, see Using Java GUI Components.

{ewl msdncd, EWGraphic, jug5s 0 /a "build.bmp"} To run Resource Wizard

- 1 From the Tools menu, choose Java Resource Wizard.
The first pane of the Java Resource Wizard dialog appears.
- 2 In the File name edit box, specify the .RCT or .RES file you want to convert to a Java file, or choose the Browse button to find it.
- 3 Choose Next.
The second pane of the Java Resource Wizard appears. To change the name of any of the classes that will be generated, double-click on the class name and enter your modifications.
- 4 Choose Finish.
A dialog box appears, displaying the names of the files generated.
- 5 Choose OK.

For each dialog or menu in the resource template, Resource Wizard creates a .JAVA file containing a class whose name is the ID of the dialog or menu. In addition, when there are dialogs in the resource template, Resource Wizard also creates a separate file for a **DialogLayout** class, which is used by the dialog class(es) to manage the layout of controls.

There's no need to edit the source code in the Resource Wizard-generated .JAVA file(s). If you want to make any changes to your menu or dialog, use the menu editor or dialog editor, re-save the resource template, and run Resource Wizard to regenerate the .JAVA file(s).

Note Each time you run Resource Wizard, it overwrites the previous version of the .JAVA file(s) it generates. As a result, any changes you make to the generated .JAVA file(s) will be lost the next time you run Resource Wizard.

For a discussion of the files that the Resource Wizard generates, see What Resource Wizard Generates.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


What Resource Wizard Generates

Dialog Templates

In the Java Abstract Window Toolkit (AWT), user-interface controls—such as buttons and scroll bars—can be arranged within various containers, such as panels, applets, windows, and dialogs.

Because dialogs are not the only kind of container, the Resource Wizard does not generate a class derived from the AWT's **Dialog** class. Instead, the Resource Wizard generates a class that adds controls to an arbitrary container to make it resemble the dialog template. In this manner, you can use Microsoft Developer Studio's dialog editor to design the appearance of not just dialogs, but also applets, panels, or windows.

This “control creator” class generated by the Resource Wizard has two methods: a constructor that takes a **Container object** as an argument, and a **CreateControls()** method. The Resource Wizard also generates a class named **DialogLayout**, which implements the AWT **LayoutManager** interface. This class is a custom layout manager which allows the position of controls to be specified in the same way that Windows resource files do. Any “control creator” classes generated by the Resource Wizard use this layout manager.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Menu Templates

In the Java AWT, menus must appear within a frame, described by the AWT's **Frame** class. The Resource Wizard generates a class that adds menus to an arbitrary frame.

This "menu creator" class generated by the Resource Wizard has two methods: a constructor that takes a **Frame** object as an argument, and a **CreateMenu()** method.

For more information on the Java AWT, see the printed book *Learn Java Now* and the section on the AWT in Sun's Java API.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Wizard-generated Dialog

The first step is to add the Resource Wizard-generated file(s) to your project, see [Adding and Removing Files from Projects](#). For the purpose of this scenario, files generated by the Resource [Wizard](#) are MyDialog.java, DialogLayout.java, and MyMenu.java.

The following steps explain how to add code that actually references the dialog classes created by the Resource [Wizard](#).

{ewl msdncd, EWGraphic, jug9s 0 /a "build.bmp"} To use the dialog class in your Java program

- 1 Open the Java source file, for example, NewApp.java, and add the following **import** statements to the beginning of the file:

```
import MyDialog;
```

- 2 Near the beginning of the applet declarations, after the declaration of the **Thread variable**, add the following line:

```
MyDialog dlg;
```

- 3 In the applet's **init()** method, remove the call to **resize()** and add the following lines:

```
dlg = new MyDialog(this);  
dlg.CreateControls();
```

- 4 The first line declares an instance of the **MyDialog** class, passing the this pointer to the constructor, specifying the applet as the container. The second line calls the **CreateControls()** method, which adds all the controls to the applet.
- 5 Comment out the call to **drawstring()** in the **paint()** method.
- 6 Just before the end of the applet declaration, add the following code:

```
public boolean action(Event evt, Object arg)  
{  
    if(evt.target instanceof Button)  
    {  
        String val = dlg.IDC_EDIT1.getText();  
        dlg.IDC_STATIC1.setText(val);  
        return true;  
    }  
    return false;  
}
```

The **action()** method responds to events generated by controls. In this example, the method checks if the event's target was a **Button object**. (Since the dialog in this example has only one **Button object**, it is not necessary to test which button was clicked.) If so, the method reads the value of the **TextField object** and uses it to set the value of the **Label object**. The names of the **TextField** and **Label** objects correspond to the IDs of the edit control and static text control, respectively, in the original dialog resource.

- 7 From the File menu, choose Save.

Note The resource template (.RCT) file you created is not part of the Java project; only the .JAVA files generated from the resource template are part of the project. Consequently, if you make modifications to the resource template—for example, by editing a menu or dialog box—the resource template does not automatically get converted the next time you build your project. You must manually run the Resource Wizard to regenerate the .JAVA files from the resource template; those .JAVA files will be automatically recompiled the next time you build the project.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Wizard-generated Menu

{ewl msdncd, EWGraphic, jug10s 0 /a "build.bmp"} To use the menu class in your Java program

- 1 In the Java source file NewApp.java, add the following **import** statement to the beginning of the file:

```
import MyMenu;
```

- 2 In the applet declaration, after the declaration of the **MyDialog** variable, add the following line:

```
MyMenu menu;  
MyFrame frame;
```

These lines declare variables for using the **MyMenu** class.

- 3 In the applet's init() method, after the lines that create the dialog, add the following lines:

```
frame = new MyFrame( "menu test" );  
frame.resize( 100, 100 );  
menu = new NewMenu( frame );  
menu.CreateMenu();  
frame.show();
```

The **MyFrame** class will be defined in the next step. These lines declare an instance of **MyFrame**, specify it as the frame for an instance of the **MyMenu** class, initialize the menus, and display the frame window.

- 4 At the end of the file, add the following code:

```
class MyFrame extends Frame  
{  
    String text = new String(" ");  
  
    MyFrame(String title)  
    {  
        super(title);  
    }  
  
    public void paint(Graphics g)  
    {  
        g.drawString(text, 10, 10);  
    }  
  
    public boolean action( Event evt, Object obj )  
    {  
        Object target = evt.target;  
        if (target instanceof MenuItem)  
        {  
            text = (String)obj;  
            repaint();  
            return true;  
        }  
        return false;  
    }  
}
```



```
}  
}
```

The **MyFrame** class extends the **Frame** class and declares a **String variable**. The override of the **paint() method** displays the value of this string. The override of the **action() method** responds to events generated within the frame. The **method** checks if the event's **target** was a **MenuItem object**. If so, the **method** sets the **text variable** to the **label** of the menu item that was chosen, which is passed in the **obj** parameter.

- 5 From the File menu, choose Save.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Seeing the Results

Build your project using the same command you used for building the sample applet. From the Build menu, choose Build.

Then, run the application. Now when the application opens its window, you'll see the menus you defined at the top of the window. The window also contains controls laid out in the manner you specified in the dialog editor.

So far, the menus and controls you've added don't do anything. In order for them to respond to user interaction, you must define event handlers. You define these methods to examine an **Event object**, determine which menu or control was chosen by the user, and take the appropriate action.

For more information on the Java AWT, see the printed book *Learn Java Now* and the section on the AWT in Sun's Java API.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Graphics Editor

The Microsoft Developer Studio graphics editor has an extensive set of tools for drawing bitmaps, icons, and cursors. However, Java currently understands only two formats for graphic images: Graphic Interface Format (.GIF) and Joint Photographic Group Format (.JPEG). Visual J++ with Microsoft Developer Studio provides the tools that allow you to create, edit, and view .GIF and .JPEG image files.

With the graphics editor, you can:

- Use the image editor window and docking toolbars.
- Customize and adjust the graphics editor workspace.
- Edit a graphic resource and draw new graphics.
- Customize colors, change palettes, and select colors.
- Create image files for Java programs.
- Convert bitmaps to image files for Java programs.

You open the Graphics Editor by creating an image file or by opening an existing image file.

Most editing procedures are the same for images, bitmaps, icons, and cursors. This section shows the procedures common to all graphical resources. The later sections detail procedures and graphic-editor capabilities specific to .JPEG and .GIF image files.

Note Many of the graphics editor's functions require a mouse or other pointing device. For keyboard shortcuts or accelerators, check the Keyboard Command Table on the Help menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Image Editor Window and Tools

You edit images in the image editor window, using the tools on the Graphics toolbar (Figure 11.1).

Figure 11.1 Image Editor Window, Graphics Toolbar, and Colors Palette

```
{ewc msdncd, EWGraphic, jug1e 0 /a "uguideIMAG.BMP"}
```

The figure shows three basic tools: the image editor window, the Graphics toolbar, and the Colors palette. Additionally, the Image menu provides useful commands, and the status bar displays helpful information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Image Editor Window

The image editor window shows two views of an image. A split bar separates the two panes. You can drag the split bar from side to side to change the relative sizes of the panes. The active pane displays a selection border, as shown in Figure 11.1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Graphics Toolbar

The Graphics toolbar has two parts, which are shown in Figure 11.1:

- The toolbar, which contains 21 tools for drawing, painting, entering text, erasing, and manipulating views.
- The option selector, which you click to select brush widths and other drawing options.

To use the Graphics toolbar, Colors palette, and option selector, click the tool, color, or option that you want.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Colors Palette

The Colors palette has two parts, which are shown in Figure 11.1:

- The color indicator, that shows the foreground and background color selectors for “screen” and “inverse” color.

Note Use this feature when creating and editing icons and cursors for a non-Java project.

- The Colors palette, which you click to select the foreground and background colors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Status Bar

The status bar, at the bottom of the frame window, displays two panes when an image editor window is open. When the pointer is over an image, the left pane shows the cursor's current position, in pixels, relative to the upper-left corner of the image. During a dragging operation, such as selecting, moving, or drawing a rectangle, the right pane shows the size, in pixels, of the affected area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Image Menu

The Image menu, which appears only when the graphics editor is active, has commands for editing images, managing color palettes, and setting image editor window options.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Managing the Graphics Editor Workspace

By adjusting the graphics editor workspace to fit your development needs, you can work more effectively and comfortably. This section describes procedures for:

- Selecting and sizing image-editor panes.
- Changing the magnification of image editor windows.
- Displaying and hiding pixel grids.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Image-Editor Panes

The image editor window typically displays an image in two panes separated by a splitter bar. One view is actual size, and the other is enlarged (the default enlargement factor is 6). The views in these two panes are updated automatically: changes you make in one pane are immediately shown in the other. The two panes make it easy for you to work on an enlarged “picture” of your image, in which you can distinguish individual pixels and, at the same time, observe the effect of your work on the actual-size view of the image.

If the image is 200 x 200 pixels or larger, however, only one pane is displayed initially. Move the splitter bar to display both panes.

You can use the two panes in other ways. For example, you might enlarge the smaller pane and use the two panes to show different regions of a large image. Click in the pane to select it.

You can change the relative sizes of the panes by positioning the pointer on the splitter bar and moving the splitter bar to the right or left. The splitter bar can move all the way to either side if you want to work on only one pane.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Changing the Magnification Factor

By default, the graphics editor displays the view in the left pane at actual size and the view in the right pane at 6 times actual size. The magnification factor is the ratio between the actual size of the image and the displayed size. The default is 6, and the range is from 1 to 8.

{ewl msdncd, EWGraphic, jug9e 0 /a "build.bmp"} To change the magnification factor

- 1 Select the image-editor pane whose magnification factor you want to change.
- 2 On the toolbar, click the Magnify tool.

The pointer changes to the Magnify tool, and magnification-factor options appear in the option selector on the Graphics toolbar. If the current magnification factor matches an option, that option is highlighted.

- 3 Click the desired magnification factor.

—or—

Press SHIFT+RIGHT ANGLE BRACKET (>) to increase the magnification factor, or press SHIFT+LEFT ANGLE BRACKET (<) to decrease the magnification factor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Displaying and Hiding the Pixel Grid

For all image-editor panes with a magnification factor of 4 or greater, you can display a grid that delimits the individual pixels in the image. For more information, see Changing the Magnification Factor.

{ewl msdncd, EWGraphic, jug10e 0 /a "build.bmp"} To display or hide the pixel grid

- 1** From the Image menu, choose Grid Settings.
The Grid Settings dialog box appears.
 - 2** Select the Pixel Grid check box to display the grid, or clear the box to hide the grid.
 - 3** Choose the OK button.
- or—
Press G to toggle the grid display.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Editing Graphical Resources

There are several editing operations involved in using the graphics editor. This section describes these graphics-editing tasks:

- Setting image properties
- Showing and hiding the Graphics toolbar
- Drawing and erasing
- Drawing lines and closed figures
- Selecting part or all of an image
- Cutting, copying, clearing, and moving selected parts of an image
- Creating a custom brush
- Flipping or resizing an image

You can also import existing images and add them to your project, and you can open files that are not part of a project for stand-alone editing.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting Image Properties

You use the Properties window to change most resource properties.

Tip By default the Properties window is hidden whenever it does not have focus. To keep the Properties window in view when it does not have focus, click the Pushpin command button in the upper-left corner of the Properties window.

{ewc msdncd, EWGraphic, jug12e 0 /a "uguide.BMP"} **To change an image's properties**

- 1 Open the image whose properties you want to change.
- 2 From the Edit menu, choose Properties to open its property page.
- 3 Change any or all of these properties on the General tab:
 - In the Width and Height boxes, modify the image's width and height (in pixels). The default value for each is 48.
If you change the dimensions of an image using the property page, the image is cropped or "blank" space is added to the right of or below the existing image.
 - In the Colors list box, select Monochrome, 16, or 256. If you have already drawn the image with a 16-color palette, selecting Monochrome causes substitutions of black and white for the colors in the image. Contrast is not always maintained: for example, adjacent areas of red and green are both converted to black.
- 4 Change any or all of the color properties on the Palette tab:
 - Double-click on a color square to display the Custom Color Selector dialog box.
 - Define the color by typing RGB or HSL values in the appropriate text boxes, or by moving the cross hairs on the color box.
 - For more information, see Changing Colors.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Showing and Hiding the Graphics Toolbar

Since many of the drawing tools are available from the keyboard, sometimes it is useful to hide the Graphics toolbar.

{ewl msdncd, EWGraphic, jug13e 0 /a "build.bmp"} To hide the Graphics toolbar

- 1 Place the mouse pointer over the toolbar area and click the right mouse button.
A pop-up menu appears.
- 2 From the pop-up menu, click Graphics.
The Graphics toolbar disappears.

{ewl msdncd, EWGraphic, jug13e 1 /a "build.bmp"} To show the Graphics toolbar

- 1 Select Toolbars from the View menu
The Toolbars dialog appears.
- 2 Select the Graphics check box.
The Graphics toolbar appears.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Freehand Drawing and Erasing

The graphics editor's freehand drawing and erasing tools all work in the same way: you select the tool and, if necessary, select foreground and background colors, as well as size and shape options. You then move the pointer to the image, and click or drag to draw and erase.

When you have selected the eraser tool, brush tool, or airbrush tool, the option selector displays that tool's options.

Tip Instead of using the eraser tool, you may find it more convenient to draw in the background color with one of the drawing tools.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Selecting and Using a Drawing Tool

The various drawing tools are easily selected using the Graphics toolbar. Figure 11.2 shows each toolbar button and its related drawing tool.

Figure 11.2 Drawing Tools in the Graphics Toolbar

```
{ewc msdn cd, EWGraphic, jug15e 0 /a "uguideDRAW.BMP"}
```

{ewl msdn cd, EWGraphic, jug15e 1 /a "build.bmp"} To select and use a drawing tool

- 1 Click a button on the Graphics toolbar:
 - The eraser tool “paints over” the image with the current background color when you press the left mouse button. When you press the right mouse button, it replaces the current foreground color with the current background color.
 - The pencil tool draws freehand in a constant width of one pixel.
 - The brush tool’s shape and size are determined by the option selector.
 - The airbrush tool randomly distributes color pixels around the center of the brush.
- 2 If necessary, select colors and a brush:
 - In the Colors palette, click the left mouse button to select a foreground color or the right mouse button to select a background color.
 - On the options selector of the Graphics toolbar, click a shape representing the brush you want to use. Your selection is highlighted.
- 3 Point to the place on the image where you want to start drawing or painting. The brush or pointer appears on the image.
- 4 Press the left mouse button (for the foreground color) or the right mouse button (for the background color), and hold it down as you draw.
- 5 Release the mouse button.

{ewl msdn cd, EWGraphic, jug15e 2 /a "build.bmp"} To change the size of the brush, airbrush, or eraser

- Press the PLUS SIGN (+) key to increase the size or the MINUS SIGN (–) key to decrease it.
–or–
Press the PERIOD (.) to choose the smallest size.
–or–
Choose a brush in the option selector.

```
{ewl msdn cd.dll, ewcright, /c"Microsoft"}
```


Drawing Lines and Closed Figures

The graphics editor tools for drawing lines and closed figures all work in the same way: you place the insertion point at one point and drag to another. For lines, these points are the endpoints. For closed figures, these points are opposite corners of a rectangle bounding the figure.

Lines are drawn in a width determined by the current brush selection, and framed figures are drawn in a width determined by the current width selection. Lines and all figures, both framed and filled, are drawn in the current foreground color if you press the left mouse button, or in the current background color if you press the right mouse button.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Drawing a Line

{ewl msdncd, EWGraphic, jug17e 0 /a "build.bmp"} To draw a line

- 1** From the toolbar, select the line tool.
- 2** If necessary, select colors. From the Colors palette, click the left button to select a foreground color or the right button to select a background color.
- 3** If necessary, select a brush. From the option selector, click a shape representing the brush you want to use. Your selection is highlighted.
- 4** Place the pointer at the line's starting point.
- 5** Drag to the line's endpoint.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Drawing a Closed Figure

The various closed-figure drawing tools are easily selected using the Graphics toolbar. Figure 11.3 shows the toolbar buttons for closed-figure drawing.

Figure 11.3 Closed-Figure Tools on the Graphics Toolbar

{ewc msdncd, EWGraphic, jug18e 0 /a "uguideCLSE.BMP"}

{ewl msdncd, EWGraphic, jug18e 1 /a "build.bmp"} To draw a closed figure

- 1 From the Graphics toolbar, select a closed-figure drawing tool:
 - The outlined-rectangle tool draws a rectangle framed with the foreground or background color.
 - The filled-rectangle tool draws a rectangle filled with the foreground or background color.
 - The outlined-round rectangle tool draws a rectangle with rounded corners framed with the foreground or background color.
 - The filled-round rectangle tool draws a rectangle with rounded corners filled with the foreground or background color.
 - The outlined-ellipse tool draws an ellipse framed with the foreground or background color.
 - The filled-ellipse tool draws an ellipse filled with the foreground or background color.
- 2 If necessary, select colors. From the Colors palette, click the left button to select a foreground color or the right button to select a background color.
- 3 If necessary, select a line width. From the option selector, click a shape representing the brush you want to use.
Your selection is highlighted.
- 4 Move the pointer to one corner of the rectangular area in which you want to draw the figure.
- 5 Drag to the diagonally opposite corner.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Selecting an Area of the Image

The selection tool defines an area of the image that you can cut, copy, clear, resize, invert, or move. You can also create a custom brush from the selection. For more information on creating a custom brush, see [Creating a Custom Brush](#).

{ewl msdncd, EWGraphic, jug19e 0 /a "build.bmp"} To select an area of the image

- 1** In the Graphics [toolbar](#), click the selection tool.
- 2** Move the [insertion point](#) to one corner of the image area that you want to select.
Cross hairs appear when the [insertion point](#) is over the image.
- 3** Drag the [insertion point](#) to the opposite corner of the area you want to select.
A rectangle shows which [pixels](#) will be selected. All [pixels](#) within the rectangle, including those “under” the rectangle, are included in the selection.
- 4** Release the mouse button.
The “selection border”—a rectangular frame—encloses the selected area. Now any operation you perform will affect only the [pixels](#) within the rectangle.

{ewl msdncd, EWGraphic, jug19e 1 /a "build.bmp"} To select the entire image

- Click the image outside of the current selection.
—or—
Press the ESC key.
—or—
Choose another tool on the [toolbar](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Cutting, Copying, Clearing, and Moving

You can perform standard editing operations—cutting, copying, clearing, and moving—with the selection, whether the selection is the entire image or just a part of it. Because the graphics editor uses the Windows Clipboard, you can transfer images between Microsoft Developer Studio and another application for Windows, such as Microsoft Paintbrush™ and Microsoft Word for Windows.

In addition, you can resize the selection, whether it includes the entire image or just a part. For more information on resizing, see [Resizing an Image](#).

{ewl msdncd, EWGraphic, jug20e 0 /a "build.bmp"} To cut the current selection and move it to the Clipboard

- From the Edit menu, choose Cut.
The original area of the selection is filled with the current background color, and the selection is now in the Clipboard.

{ewl msdncd, EWGraphic, jug20e 1 /a "build.bmp"} To clear the current selection without moving it to the Clipboard

- From the Edit menu, choose Clear.
The original area of the selection is filled with the current background color.

{ewl msdncd, EWGraphic, jug20e 2 /a "build.bmp"} To paste the Clipboard contents into the image

- 1 From the Edit menu, choose Paste.
The Clipboard contents, surrounded by the selection border, appear in the upper-left corner of the pane.
- 2 Position the pointer within the selection border and drag the contents to the desired location on the image.
- 3 To anchor the selection at its new location, click outside of the selection border.
—or—
Choose a new tool.

{ewl msdncd, EWGraphic, jug20e 3 /a "build.bmp"} To move the selection

- 1 Position the pointer inside the selection border or anywhere on it, except the sizing handles.
- 2 Drag the selection to its new location.
The original area of the selection is filled with the current background color.
- 3 To anchor the selection at its new location, click outside the selection border.
—or—
Choose a new tool.

{ewl msdncd, EWGraphic, jug20e 4 /a "build.bmp"} To copy the selection

- 1 Position the pointer inside the selection border or anywhere on it, except the sizing handles.
- 2 Hold down the CTRL key as you drag the selection to a new location.
The area of the original selection is unchanged.
- 3 To copy the selection at its current location, click outside the selection cursor.
—or—

Choose a new tool.

{ewl msdncd, EWGraphic, jug20e 5 /a "build.bmp"} To draw with the selection

- 1 Position the pointer inside the selection border or anywhere on it, except the sizing handles.
- 2 Hold down the SHIFT key as you drag the selection.

Copies of the selection are left along the dragging path. The more slowly you drag, the more copies are made.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Flipping the Selection

{ewl msdncd, EWGraphic, jug21e 0 /a "build.bmp"} To flip the selection along the horizontal axis

- From the Image menu, choose Flip Horizontal.

{ewl msdncd, EWGraphic, jug21e 1 /a "build.bmp"} To flip the selection along the vertical axis

- From the Image menu, choose Flip Vertical.

{ewl msdncd, EWGraphic, jug21e 2 /a "build.bmp"} To rotate the selection 90°

- From the Image menu, choose Rotate 90°.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a Custom Brush

A custom brush is a rectangular portion of an image that you “pick up” and use like one of the graphics editor’s ready-made brushes. All operations you can perform on a selection, you can perform on a custom brush as well.

{ewl msdncd, EWGraphic, jug22e 0 /a "build.bmp"} To create a custom brush

- 1 Select the part of the image that you want to use for a brush. For more information, see Selecting and Using a Drawing Tool.
- 2 Press CTRL+B.

Pixels in a custom brush that match the current background color are normally “transparent”—they do not paint over the existing image. You can change this behavior so that background-color pixels paint over the existing image.

You can use the custom brush like a “stamp” or a “stencil” to create a variety of special effects.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using a Custom Brush

{ewl msdncd, EWGraphic, jug23e 0 /a "build.bmp"} To draw custom brush shapes in the background color

- 1 Select an opaque or transparent background. For more information, see [Choosing Opaque and Transparent Backgrounds](#).
- 2 Set the background color to the color want.
- 3 Position the custom brush where you want.
- 4 Press the right mouse button.

Any opaque regions of the custom brush are drawn in the background color.

{ewl msdncd, EWGraphic, jug23e 1 /a "build.bmp"} To double or halve the custom brush size

- Press the PLUS SIGN (+) key to double the brush size or the MINUS SIGN (–) key to halve it.

{ewl msdncd, EWGraphic, jug23e 2 /a "build.bmp"} To cancel the custom brush

- Press ESC.
–or–
Choose another drawing tool.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Resizing an Image

The behavior of the graphics editor while resizing an image depends on whether the selection includes the entire image or just part of it:

- When the selection includes only part of the image, Microsoft Developer Studio shrinks the selection by deleting rows or columns of pixels and filling the vacated regions with the current background color, or it stretches the selection by duplicating rows or columns of pixels.
- When the selection includes the entire image, Developer Studio either shrinks and stretches the image, or crops and extends it.

There are two mechanisms for resizing an image: the sizing handles and the property page. You can drag the sizing handles to change the size of all or part of an image. You can drag sizing handles that are solid, like those on the lower-right corner and the midpoints of the right and bottom sides of the images. You cannot drag handles that are hollow. You can use the property page to resize only the entire image, not a selected part.

Note If you have the Tile Grid option selected (see Grid Settings command on the Image menu), then resizing snaps to the next tile grid line. If only the Pixel Grid option is selected, resizing snaps to the next available pixel. Usually, only the Pixel Grid option is selected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Changing the Size of an Image

{ewl msdncd, EWGraphic, jug25e 0 /a "build.bmp"} To resize an entire image using the property page

- 1 From the Edit menu, choose Properties to open the property page.
- 2 In the Width and Height boxes, type the dimensions that you want.
If you are increasing the size of the image, the graphics editor extends the image to the right downward, or both, and fills the new region with the current background color. The image is not stretched.
If you are decreasing the size of the image, the graphics editor crops the image on the right or bottom edge, or both.

You can use the Width and Height properties to resize only the entire image, not to resize a partial selection.

{ewl msdncd, EWGraphic, jug25e 1 /a "build.bmp"} To crop or extend an entire image

- 1 Select the entire image.
If part of the image is currently selected, and you want to select the entire image, click anywhere on the image outside the current selection border.
—or—
Press ESC.
—or—
Choose another drawing tool.
- 2 Drag a sizing handle until the image is the desired size.

Normally, the graphics editor crops or enlarges an image when you resize it by moving a sizing handle. If you hold down the SHIFT key as you move a sizing handle, the graphics editor shrinks or stretches the image.

{ewl msdncd, EWGraphic, jug25e 2 /a "build.bmp"} To shrink or stretch an entire image

- 1 Select the entire image.
If a part of the image is currently selected and you want to select the entire image, click anywhere on the image outside the current selection border.
—or—
Press ESC.
—or—
Choose another drawing tool.
- 2 Hold down the SHIFT key and drag a sizing handle until the image is the desired size.

{ewl msdncd, EWGraphic, jug25e 3 /a "build.bmp"} To shrink or stretch part of an image

- 1 Select the part of the image you want to resize. For more information, see Selecting an Area of the Image.
- 2 Drag one of the sizing handles until the selection is the desired size.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Working With Colors in the Graphics Editor

The graphics editor comes equipped with many features specifically to help with the handling and customizing of colors. You can:

- Set foreground and background colors, and choose opaque and transparent backgrounds.
- Fill an area of an image with a color or quickly pick up a color from the image to use it elsewhere.
- Invert the colors in a selection.
- Customize or change the colors.
- Save and load different color palettes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Selecting Foreground and Background Colors

Except for the eraser, the tools on the Graphics toolbar draw with the current foreground or background color when you press the left or right mouse button, respectively.

{ewl msdncd, EWGraphic, jug27e 0 /a "build.bmp"} To select a foreground color

- With the left mouse button, click the color you want on the Colors palette.

{ewl msdncd, EWGraphic, jug27e 1 /a "build.bmp"} To select a background color

- With the right mouse button, click the color you want on the Colors palette.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Filling Bounded Areas

The graphics editor provides the fill (or “paint-bucket”) tool for filling any enclosed image area with the current drawing color or the current background color.

{ewl msdncd, EWGraphic, jug28e 0 /a "build.bmp"} To use the fill tool

- 1** From the Graphics toolbar, choose the fill tool.
- 2** If necessary, choose drawing colors. From the Colors palette, click the left button to select a foreground color or the right button to select a background color.
- 3** Move the fill tool to the area you want to fill.
- 4** Click the left or right mouse button to fill with the foreground color or the background color, respectively.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Picking Up Colors

The color-pickup tool makes any color on the image the current foreground color or background color, depending on whether you press the left or the right mouse button. To cancel the color pickup tool, choose another tool or press ESC.

{ewl msdncd, EWGraphic, jug29e 0 /a "build.bmp"} To pick up a color

- 1** From the Graphics toolbar, select the color-pickup tool.
The pointer changes to the “eyedropper.”
- 2** Select the color you want to pick up from the Colors palette or from the Palette tab of the property page.
After you pick up a color, the graphics editor reactivates the most recently used tool.
- 3** Draw using the left mouse button for the foreground color, or the right mouse button for the background color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choosing Opaque and Transparent Backgrounds

When you move or copy a selection from an image, any pixels in the selection that match the current background color are by default “transparent”—they do not obscure pixels in the target location. A custom brush behaves in the same way. For more information on custom brushes, see Creating a Custom Brush.

{ewl msdncd, EWGraphic, jug30e 0 /a "build.bmp"} To toggle the background-color transparency

- In the Graphics toolbar option selector, click the Opaque background button to have the existing image obscured by all parts of the selection.
—or—
Click the Transparent background button to have the existing image show through those parts of the selection that match the current background color.

You can change the background color while a selection is already in effect to change which parts of the image are transparent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Inverting Colors in the Current Selection

So that you can tell how an image would appear with inverted colors, the graphics editor provides a convenient way to invert colors in the selected part of the image.

{ewl msdncd, EWGraphic, jug31e 0 /a "build.bmp"} To invert colors in the current selection

- From the Image menu, choose Invert Colors.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Changing Colors

The graphics editor's Colors palette initially displays 24 "ready-made" colors: 16 standard colors and 8 dithered colors. In addition to the ready-made colors, you can create your own custom colors. Colors palette selections can be saved on disk and individually reloaded as needed. The "most recently used" Colors palette definition that is saved in the Registry is automatically loaded the next time you start Developer Studio.

The Palette tab in the Properties window displays up to 256 colors. Changing any of the colors on the Palette tab will immediately change the corresponding color in the image. The colors on the Palette tab are always solid colors and can indicate any color your video card is capable of displaying.

{ewl msdncd, EWGraphic, jug32e 0 /a "build.bmp"} To change colors on the Colors palette or Palette tab

- 1 From the Image menu, choose Adjust Colors.
—or—
Double-click one of the color squares on the Colors palette.
—or—
Double-click one of the color squares on the Palette tab of the Bitmap Properties page.
The Custom Color Selector dialog box (Figure 11.4) appears.

Figure 11.4 Custom Color Selector Dialog Box

{ewc msdncd, EWGraphic, jug32e 1 /a "uguideCUST.BMP"}

- 2 Define the color by typing RGB or HSL values in the appropriate text boxes, or by moving the cross hairs on the color box.
- 3 Set the luminance by moving the slider on the luminance bar.
Many custom colors are dithered. If you want the solid color closest to the dithered color, double-click the Color preview window. (If you later decide you want the dithered color, move the slider or the cross hairs again to restore the dithering.)
- 4 Choose OK to add the new color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Saving and Loading Colors Palettes

You use commands on the Image menu to save or load a palette.

{ewl msdncd, EWGraphic, jug33e 0 /a "build.bmp"} To save a custom Colors palette

- 1 From the Image menu, choose Save Palette.
- 2 Use the Save Palette Colors dialog box to navigate directories and type a filename.

{ewl msdncd, EWGraphic, jug33e 1 /a "build.bmp"} To load a custom Colors palette

- 1 From the Image menu, choose Load Palette.
- 2 Use the Load Palette Colors dialog box to navigate directories and choose a filename.

Tip Since the graphics editor has no means to restore the default Colors palette, save the default Colors palette under a name such as STANDARD.PAL or DEFAULT.PAL so that you can easily restore the default settings.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating an Image File

You can create new .GIF and .JPEG files using the graphics editor.

{ewl msdncd, EWGraphic, jug34e 0 /a "build.bmp"} To create a new .GIF or .JPEG image file

- 1** From the File menu, choose New.
The New dialog box appears.
- 2** Select Bitmap File, and then click OK.
The graphics editor opens with an empty grid.
- 3** Create your image using the graphics editor's tools and resources. For more information on using the graphics editor, see Using the Image Editor Window and Tools and Editing Graphical Resources.
- 4** To save your finished image, select Save from the File menu.
—or—
Click Save on the File toolbar.
The Save As dialog box appears.
- 5** Navigate to the directory where you want to save the file, and type the name of your new image file. Give it an extension of either .GIF or .JPEG, depending on which format you want to use in your program.
- 6** Choose OK.

To add a .GIF or .JPEG file to your project, pull down the Insert menu and choose Files into Project.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Converting a Bitmap to an Image File

If you have bitmap files that you would like to use with your Java program, the graphics editor can help you convert these files to the required .GIF and .JPEG formats.

{ewl msdncd, EWGraphic, jug35e 0 /a "build.bmp"} To convert an existing bitmap file (.BMP) to a .GIF or .JPEG image file

- 1 From the File menu, choose Open.
The Open dialog box appears.
- 2 Navigate to the directory that contains the bitmap you want.
- 3 Select Image Files from the Type of Files drop-down list box.
- 4 Enter the name of the bitmap file in the File Name edit box, choose Open.
—or—
Select the file from the list of files displayed, choose Open.
—or—
Double-click on the file name.
The bitmap appears in the image window of the graphics editor.
- 5 If you plan to edit the bitmap, this would be a good time to do it. Edit the bitmap using the graphics editor's tools and resources. For more information on using the graphics editor, see Using the Image Editor Window and Tools and Editing Graphical Resources.
- 6 To save your finished image, select Save As from the File menu.
The Save As dialog box appears.
- 7 Navigate to the directory where you want to save the file, and type the name of your new image file. Give it an extension of either .GIF or .JPEG depending on which format you want to use in your program.
- 8 Choose OK.

To add a .GIF or .JPEG file to your project, pull down the Insert menu and choose Files into Project.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Debugger

Microsoft Developer Studio provides an integrated debugger to help locate bugs in a Java application or applet. Using the debugger, you can:

- Use multiple debug windows to display the call stack, variables, and bytecode.
- Control and manage breakpoints.
- Control threads in multithreaded programs.

Using the debugger, you can control the execution of your program and examine the program state at selected points through multiple windows and dialog boxes. You can set breakpoints to halt execution at critical locations.

When running a program with the debugger, you can single-step to observe the effects of your code. You can choose to enter a method, using the Step Into command, or step past a method call, using the Step Over command. You can exit a called method and return to the calling statement using the Step Out command.

When you end a debugging session, the project retains any breakpoints you have set. When you reopen the project, the debugger restores the breakpoints in the proper locations. When the program is halted at a breakpoint, you can use the debug windows and dialog boxes to examine the state of your program.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Debugger Interface Components

To use the debugger, you use interface components including menus, windows, dialog boxes, and spreadsheet fields. Drag-and-drop functionality is available for moving debug information between components. For more information, see:

- Menu commands
- Windows
- Dialog boxes
- Pop-up menus
- Spreadsheet fields
- Drag-and-drop

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Debugger Menu Items

Commands for debugging can be found on the Debug menu, the Build menu, the View menu, and the Edit menu.

The Debug menu appears in the menu bar while the debugger is running (even if it is stopped at a breakpoint). From the Debug menu, you can control program execution and access the QuickWatch window. When the debugger is not running, the Debug menu is replaced by the Build menu. The Build menu contains a command called Debug, which contains a subset of the commands on the full Debug menu. These commands start debugging (Go, Step Into, and Run To Cursor). For a description of the Debug menu commands and the Debug commands on the Build menu, see Controlling Program Execution.

The View menu contains commands that display the various debugger windows, such as the Variables window and the Call Stack window. For more information about the debugger windows, see Viewing and Modifying Variables and Expressions. For details, see Using Breakpoints.

The Edit menu contains a command to open the Breakpoint dialog to insert or edit a breakpoint from which you can insert, remove, enable or disable breakpoints.

A pop-up menu appears whenever you click the right mouse button in a debugger window. This menu provides commonly used commands applicable to that window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Debugger Windows

Several specialized windows display debugging information for your program. When you are debugging, you can access these windows using the View menu. (In addition to windows, the debugger uses dialog boxes to display debugging information.)

Table 12.1 lists the debugger windows and describes the information they display.

Table 12.1 Debugger Windows

Win dow	Displays
Outp ut	Information about the build process , including any compile or build-tool errors, as well as output from thread termination codes.
Watc h	Names and values of variables and expressions.
Varia bles	Information about variables used in the current and previous statements, variables local to the current method (in the Locals tab), and the object pointed to by this (in the This tab).
Call Stac k	Stack of all method calls that have not returned.
Disa sse	Virtual Machine

mbly bytecode
derived from
disassembly
of the
compiled
program.

Debugger windows can be docked or floating. For information on docked and floating windows, see [Working with Docking Tool Windows](#).

When a window is in floating mode, you can resize or minimize it to increase the visibility of other windows. You can copy information from any [debugger](#) window. You can print information only from the Output window.

Tip To set formatting and other options for these windows, use the Debug tab in the Options [dialog box](#) (accessed from the Tools menu).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Pop-up Menus

Each debugger window has a pop-up menu, which contains frequently used commands for that window.

{ewl msdncd, EWGraphic, jug4f 0 /a "build.bmp"} To display the pop-up menu for a window

- Click the right mouse button inside the window.

For example, if you click the right mouse button in the Variables window, you see a pop-up menu with several formatting options. If you click the right mouse button in a source/text window, you see a pop-up menu with several commands, including Insert/Remove Breakpoint. If you click a variable *Var* within a source window while you are debugging, the pop-up menu includes the command QuickWatch *Var*, which displays the variable *Var* in the QuickWatch dialog box.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Debugger Dialog Boxes

In addition to windows, the debugger uses a number of dialog boxes to manipulate breakpoints, variables, threads, and exceptions. You can access the Breakpoints dialog box using the Breakpoints command on the Edit menu. You can access the other dialog boxes using commands from the Debug menu.

Table 12.2 lists the debugger dialog boxes and describes the information they display.

Table 12.2 Debugger Dialog Boxes

	Dia log bo x
Breakpoints	List of all <u>breakpoints</u> assigned to your project. Use the tabs in the Breakpoints <u>dialog box</u> to create new <u>breakpoints</u> of various types.
Exceptions	System and user-defined exceptions for your project. Use the Exceptions <u>dialog box</u> to control how the <u>debugger</u> handles exceptions.
QuickWatch	A <u>variable</u> or expression. Use QuickWatch to quickly <u>view</u> or modify a <u>variable</u> or expression or to add it to the Watch window.
Threads	Application threads available for debugging. Use the

Threads
[dialog box](#) to
suspend and
resume threads
and to set
[focus](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Spreadsheet Fields

The debugger interface uses spreadsheet fields that have an interface similar to that of Microsoft Excel. These spreadsheet fields appear in the Watch window, the Variables window, and the QuickWatch dialog box.

When working with these fields, you can autosize a column to fit its contents by double-clicking the divider. You can size a column manually by dragging the divider at the right edge of the column.

Note Rows fit the current font and cannot be resized. To change the font size, use the Format tab of the Options dialog box, accessed from the Tools menu.

Spreadsheet fields contain controls for easy viewing of array, object, structure, and pointer variables. These variables are marked with a box containing a plus sign (+) in the Name column. You can expand the variable by clicking the + box, which opens into a tree that may contain additional boxes.

If the variable is a pointer, the branch immediately below the pointer contains the value pointed to. If the variable is an array, object, or structure, the branch below the variable contains the component elements or members. When a variable is expanded, the box in the Name column contains a minus sign (–). You can collapse an expanded variable by clicking the – box. As an alternative, you can expand a variable by selecting it and pressing the PLUS SIGN OR RIGHT ARROW key. You can collapse a variable by selecting it and pressing the MINUS SIGN OR LEFT ARROW key.

Scalar variables, which have no components to expand, do not have boxes in the Name column.

{ewl msdncd, EWGraphic, jug6f 0 /a "build.bmp"} To select a spreadsheet cell for editing

- 1 Select the spreadsheet.
- 2 Use the UP ARROW and DOWN ARROW keys to move to the correct line.
- 3 To select the cell, press TAB to advance the selection to the next editable cell, or press SHIFT+TAB to move the selection back to the previous editable cell.

The window where the spreadsheet field is located determines which cells are editable. In the Variables window, you can edit the cells in the Value column. In the Watch window, you can edit the cells in the Name and Value columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dragging and Dropping Debugger Information

You can move information between debugger windows using drag-and-drop or cut-and-paste features. When you select information and drag it with the mouse, the mouse pointer changes. A small, gray rectangle appears at the base of the arrow to indicate that the information can be dropped. If you move the mouse pointer across a window or area that cannot accept a drop, the mouse pointer temporarily changes into the “No” symbol—a circle with a slash through it.

The debugger interface supports intelligent drag-and-drop. The result of a drag-and-drop operation depends, in part, on the location where the drop takes place.

For example, you can drag a variable from the Variables window to the Watch window. This action puts the variable information into the Watch window, where it is updated each time the Watch window updates. If you drag the variable to a text window, instead, the variable information is converted into text. If you drag the variable to the Disassembly window, the window scrolls to display relevant instruction(s).

If you expand an object (**Obj**, for example) in the Variables window, you can drag a member of that object (such as **Obj.child**) to the Watch window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Controlling Program Execution

To start debugging, choose the Go, Step Into, or Run To Cursor command under Debug on the Build menu. Table 12.3 lists the Build Debug menu commands and their actions.

Table 12.3 Build Menu Debug Commands

Co mm and	Action
Go (F5)	Executes code from the current statement until a breakpoint is reached or the end of the program is reached. (Equivalent to the Go button on the toolbar.)
Ste p Into (F11)	Single-steps through instructions in the program, and enters each method call that is encountered.
Ru n to Cur sor (CT RL+ F10)	Executes the program as far as the line that contains the insertion point . This is equivalent to setting a temporary breakpoint at the insertion point location.

When you begin debugging, the Debug menu appears, replacing the Build menu on the menu bar. You can then control program execution using the commands listed in Table 12.4.

Table 12.4 Debug Menu Commands that Control Program Execution

Co mm and	Action
Go	Executes code from the current

statement until a breakpoint is reached or the end of the program is reached.

(Equivalent to the Go button on the Standard toolbar.) When the Debug menu is not available, you can choose from Go from the Debug submenu of the Build menu.

Res Resets
Start execution to the first line of the program. This command reloads the program into memory, and discards the current values of all variables ([breakpoints](#) and [watch expressions](#) still apply). It automatically halts at the **main()** [method](#) in Java applications and the **init()** [method](#) in Java applets..

Stop Terminates the debugging
Debug [session](#) and returns to a normal editing
g [session](#).

Break Halts the program at its current

location.

Ste Single-steps
p through the
Into code in the
program, and
enters each
[method](#) call
that is
encountered.
When the
Debug menu is
not available,
you can
choose Step
Into from the
Debug
submenu of
the Build
menu.

Ste Single-steps
p through the
Ov code in the
er program. If this
command is
used when you
reach a
[method](#) call,
the [method](#) is
executed
without
stepping
through the
[method](#) code.

Ste Executes the
p program out of
Out a [method](#)
call, and stops
on the
instruction
immediately
following the
call to the
[method](#).
Using this
command, you
can quickly
finish
executing the
current
[method](#) after
determining
that a bug is
not present in

the [method](#).

Ru Executes the
n to program as far
Cur as the line that
sor contains the
[insertion
point](#). This
command is
equivalent to
setting a
temporary
breakpoint at
the [insertion
point](#) location.
When the
Debug menu is
not available,
you can
choose Run To
Cursor from
the Debug
submenu of
the Build
menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Running to a Location

{ewl msdncd, EWGraphic, jug9f 0 /a "build.bmp"} To run until a breakpoint is reached

- From the Debug menu, choose Go.

{ewl msdncd, EWGraphic, jug9f 1 /a "build.bmp"} To run to the cursor

- 1 Open a source file, and move the insertion point to the location where you want the debugger to break.
- 2 From the Debug menu, choose Run To Cursor.

The Run To Cursor command also works in the Call Stack window, the Disassembly window, and the Find box on the standard toolbar.

{ewl msdncd, EWGraphic, jug9f 2 /a "build.bmp"} To run to the cursor location in object code

- 1 In the Disassembly window, move the insertion point to the location where you want the debugger to break.
- 2 From the Debug menu, choose Run To Cursor.

{ewl msdncd, EWGraphic, jug9f 3 /a "build.bmp"} To run to the cursor location in the call stack

- 1 In the Call Stack window, select the method name.
- 2 From the Debug menu, choose Run To Cursor.

{ewl msdncd, EWGraphic, jug9f 4 /a "build.bmp"} To run to a specified method

- 1 In the Find box on the standard toolbar, type the method name.
- 2 From the Debug menu, choose Run To Cursor.

Tip You can use the Run To Cursor command to return to an earlier statement to retest your application, using different values for variables.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Stepping Into Methods

Once your program has stopped at a breakpoint, you can step through the code one statement at a time using the Step Into command from the Debug menu or the Step Into button on the Debug toolbar.

{ewl msdncd, EWGraphic, jug10f 0 /a "build.bmp"} To run the program and execute the next statement (Step Into)

- 1 While the program is paused at a breakpoint, choose Step Into from the Debug menu.
The debugger executes the next statement, then pauses execution. If the next statement is a method call, the debugger steps into that method, then pauses execution at the beginning of the method.
- 2 Repeat step 1 to continue executing the program one statement at a time.

If you use this technique to step into a nested method call, the debugger steps into the most deeply nested method. Consider, for example, the following line of code:

```
Obj.Fun (Obj.Fun2 ( ) ) ;
```

If you use the Step Into command, the debugger steps into the method **Obj.Fun2()**, then pauses. If you use the Step Into Specific Function command instead, you can control which method the debugger steps into. In the following example, you can use the Step Into Specific Function command to step into **Obj.Fun()** without first stepping through **Obj.Fun2()**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Stepping Over or Out of Routines

You can step through your program one statement at a time in a chosen method, starting from a breakpoint, without entering any other methods, by using the Step Over command. You can also exit from a method immediately and return to the line where the method was called by using the Step Out command.

{ewl msdncd, EWGraphic, jug11f 0 /a "build.bmp"} To run the next statement in the current method

- 1 Open a source file, and set a breakpoint in the method.
- 2 From the Debug menu, choose Go.
When the program comes to the breakpoint, the debugger pauses.
- 3 From the Debug menu, choose Step Over.
The debugger executes the next method, but pauses after the method returns.
- 4 Repeat step 3 to continue executing the program, one statement at a time.

You can stop the debugger after it has executed the return statement in a method. It stops on the line following the method call.

{ewl msdncd, EWGraphic, jug11f 1 /a "build.bmp"} To run the program and stop execution after the current method returns to the calling method

- 1 Open a source file, and set a breakpoint in the method.
- 2 From the Debug menu, choose Go.
When the program comes to the breakpoint, the debugger pauses.
- 3 From the Debug menu, choose Step Out.
The debugger continues until it has completed execution of the return from the current method, then pauses.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Interrupting Your Program

There may be times when you cannot set a breakpoint to halt the program, such as when your program encounters an infinite loop. In such cases, you can interrupt your program by choosing Break on the Debug menu. This action returns control to Microsoft Developer Studio and opens the Disassembly window. You can then use the Go, Step Into, or Step Over commands to regain control of your program.

Note If you interrupt execution while Windows or other Virtual Machine bytecode is running, the results can be unpredictable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Breakpoints

Breakpoints tell the debugger where or when to break execution of a program. When the program is halted at a breakpoint, you can examine the state of your program, step through your code, and evaluate expressions using the debugger windows.

The debugger supports Location breakpoints that halt the debugger at a specified location.

The Breakpoints dialog box displays a list of all breakpoints set in the project. You can use this dialog box to set, remove, disable, and enable breakpoints. When you close a project, the debugger saves all breakpoints you have set as part of the project information. The next time you open the project, the breakpoints remain as you left them.

The debugger also provides quick methods for setting location breakpoints without using the Breakpoints dialog box.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Quick Methods for Location Breakpoints

With the debugger, you can set location breakpoints:

- On a specific line of source code.
- At the beginning or the return point of a method.
- At a label.

You can set or remove any of these location breakpoints without using the Breakpoints dialog box. You can disable and enable breakpoints on a source-code line or in the Disassembly or Call Stack window.

{ewl msdncd, EWGraphic, jug14f 0 /a "build.bmp"} To set a breakpoint at a source-code line

- 1 In a source window, move the insertion point to the line where you want the program to break.
- 2 Choose the Insert/Remove Breakpoint toolbar button.
A red dot appears in the left margin, indicating that the breakpoint is set.

Note If you want to set a breakpoint on a source statement extending across three or more lines, you must set the breakpoint on the first or last line of the statement.

{ewl msdncd, EWGraphic, jug14f 1 /a "build.bmp"} To set a breakpoint at the beginning of a method

- 1 In the Find box on the Standard toolbar, type the method name, and press ENTER.
- 2 Choose the Insert/Remove Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Insert/Remove Breakpoint from the pop-up menu.
In the source code, a red dot appears in the left margin at the beginning of the method, indicating that the breakpoint is set.

{ewl msdncd, EWGraphic, jug14f 2 /a "build.bmp"} To set a breakpoint at the return point of a method

- 1 In the Call Stack window, move the insertion point to the method where you want the program to break.
- 2 Choose the Insert/Remove Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Insert/Remove Breakpoint from the pop-up menu.
A red dot appears in the left margin, indicating that the breakpoint is set.

{ewl msdncd, EWGraphic, jug14f 3 /a "build.bmp"} To set a breakpoint at a label

- 1 In the Find box on the Standard toolbar, type the name of the label, and press ENTER.
- 2 Choose the Insert/Remove Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Insert/Remove Breakpoint from the pop-up menu.

A red dot appears in the left margin at the line containing the label, indicating that the breakpoint is set.

{ewl msdncd, EWGraphic, jug14f 4 /a "build.bmp"} To disable a breakpoint

- 1 In a source window, or in the Call Stack or Disassembly window, move the insertion point to the line containing the breakpoint you want to disable.
- 2 Choose the Enable/Disable Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Disable Breakpoint from the pop-up menu.
The red dot in the left margin changes to a hollow circle.

{ewl msdncd, EWGraphic, jug14f 5 /a "build.bmp"} To disable all breakpoints

- Choose the Disable All Breakpoints toolbar button.
The red dots in the left margin change to hollow circles.

{ewl msdncd, EWGraphic, jug14f 6 /a "build.bmp"} To enable a breakpoint

- 1 In a source window, or in the Call Stack or Disassembly window, move the insertion point to the line containing the breakpoint you want to enable.
- 2 Choose the Enable/Disable Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Enable Breakpoint from the pop-up menu.
The hollow circle in the left margin changes to a red dot.

Note If you set more than one breakpoint on a line, and some breakpoints are disabled while others are enabled, a gray dot appears in the left margin. The first time you choose the Enable/Disable Breakpoint toolbar button, all breakpoints on the line become disabled, and the gray dot changes to a hollow circle. If you choose the Enable/Disable Breakpoint button again, all breakpoints on the line become enabled, and the hollow circle changes to a red dot.

{ewl msdncd, EWGraphic, jug14f 7 /a "build.bmp"} To remove a breakpoint

- 1 In a source window, or in the Call Stack or Disassembly window, move the insertion point to the line containing the breakpoint you want to remove.
- 2 Choose the Insert/Remove Breakpoint toolbar button.
—or—
Click the right mouse button, and choose Remove Breakpoint from the pop-up menu.
The red dot in the left margin disappears.

Note If a line contains enabled and disabled breakpoints, the Insert/Remove Breakpoint button removes all enabled breakpoints. Disabled breakpoints are not affected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Breakpoints Dialog Box

Using the Breakpoints dialog box, accessed by the Breakpoints command on the Edit menu, you can set, remove, disable, enable, or view location breakpoints.

Location breakpoints are set at a specific line of source code or the start of a method. They break execution of the program when the location counter reaches that point in the program.

For more information see, the Location Breakpoints Tab.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Breakpoints List

The Breakpoints dialog box contains a list of all breakpoints currently set in your program. You can use this list to examine all breakpoints in your program, to disable breakpoints, or to enable breakpoints that you previously disabled. You can also use the list to remove (delete) a breakpoint.

{ewl msdncd, EWGraphic, jug16f 0 /a "build.bmp"} To view the list of current breakpoints

- 1 From the Edit menu, choose Breakpoints.
The Breakpoints dialog box appears.
- 2 Use the scroll bars to move up or down the Breakpoints list.

{ewl msdncd, EWGraphic, jug16f 1 /a "build.bmp"} To disable a breakpoint

- 1 In the Breakpoints dialog box, find the breakpoint in the Breakpoints list.
- 2 Clear the check box corresponding to the breakpoint that you want to disable.
- 3 Choose OK.
The red dot in the left margin changes to a hollow circle.

{ewl msdncd, EWGraphic, jug16f 2 /a "build.bmp"} To enable a breakpoint

- 1 In the Breakpoints dialog box, find the breakpoint in the Breakpoints list.
- 2 Select the empty check box corresponding to the breakpoint that you want to enable.
- 3 Choose OK.
The red dot in the left margin changes to a hollow circle.

Tip You can also use the SPACEBAR to toggle the state of a breakpoint in the Breakpoints list.

{ewl msdncd, EWGraphic, jug16f 3 /a "build.bmp"} To remove a breakpoint

- 1 In the Breakpoints dialog box, select one or more breakpoints in the Breakpoints list.
- 2 Choose the Remove button.
—or—
Press the DELETE key.
- 3 Choose OK.

When you select a breakpoint in the Breakpoints list, the breakpoint information automatically appears in the text box of the Location. You can edit the breakpoint using the procedures described in the topic The Location Breakpoints Tab.

Note You can use the Edit Code button to navigate to the source or object code where the location breakpoint is set.

{ewl msdncd, EWGraphic, jug16f 4 /a "build.bmp"} To view the source code where a breakpoint is set

- 1 In the Breakpoints list, select a line-number breakpoint.
- 2 Choose the Edit Code button.

This action takes you to the source code for a breakpoint set at a line number, or to the disassembled bytecode for a breakpoint.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Location Breakpoints Tab

You can use the Location tab in the Breakpoints dialog box to set a location breakpoint:

- On a specific line of source code.
- At the start of a method.

Note Except where noted, the following procedures work only within the current context (method or source file).

{ewl msdncd, EWGraphic, jug17f 0 /a "build.bmp"} To set a breakpoint at the current location

- 1 From the Edit menu, choose Breakpoints.
The Breakpoints dialog box and the location tab appears.
- 2 Click the right arrow button next to the Break At text box.
- 3 From the menu that appears, click on the current line number.
- 4 Choose OK to set the breakpoint.

Note If you want to set a breakpoint on a source statement extending across three or more lines, you must set the breakpoint on the first or last line of the statement.

{ewl msdncd, EWGraphic, jug17f 1 /a "build.bmp"} To set a breakpoint at another location

- 1 On the Location tab, type a source-code line number (if the current location is in a source file) directly into the Break At text box. For source locations, type a period immediately before the line number.
- 2 Choose OK to set the breakpoint.

Note If you want to set a breakpoint on a source statement extending across three or more lines, you must set the breakpoint on the first or last line of the statement.

{ewl msdncd, EWGraphic, jug17f 2 /a "build.bmp"} To set a breakpoint at the beginning of the current method

- 1 In the Location tab, click on the drop-down menu next to the Break At box.
- 2 From the menu choose the current method name that appears.
- 3 Choose OK to set the breakpoint.

Note If the debugger is halted in disassembled object code, rather than source code, this option is not available.

{ewl msdncd, EWGraphic, jug17f 3 /a "build.bmp"} To set a breakpoint at the beginning of another method

- 1 In the Location tab, type the method name directly into the Break At text box.
- 2 Choose OK to set the breakpoint.

{ewl msdncd, EWGraphic, jug17f 4 /a "build.bmp"} To edit a location breakpoint

- 1** In the Location tab, select the location breakpoint in the breakpoints list.
- 2** Edit the location that appears in the Break At text box.
- 3** Choose OK to set the breakpoint.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Viewing and Modifying Variables and Expressions

The debugger provides several ways to view the value of a variable or expression:

- DataTips Pop-up Information
- QuickWatch
- The Watch window
- The Variables window

It also provides several ways to modify the value of a variable:

- QuickWatch
- The Watch window
- The Variables window

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using DataTips Pop-up Information

The easiest way to see the value of a variable or expression when the debugger is stopped at a breakpoint is to use DataTips pop-up information.

You can view a DataTips pop-up information box for any variable or expression that appears in a source window and is within the current scope. To see a pop-up box for a variable, place the mouse pointer over the variable. To see a pop-up box for an expression, select the expression.

DataTips pop-up information is not available for invalid expressions, such as a division by zero. If you select an expression such as $1/0$, no pop-up information box appears.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using QuickWatch

You can use QuickWatch to quickly examine the value of a variable or expression. You can also use QuickWatch to modify the value of a variable or to add a variable or expression to the Watch window.

The QuickWatch dialog box contains a text box, where you can type an expression or variable name, and a spreadsheet field that displays the current value of the variable or expression that you specified.

The Current Value spreadsheet field displays only one variable or expression at a time. If you type a new variable or expression in the text box and press ENTER, the previous variable or expression in the Current Value field is replaced.

If you type a scalar variable or expression in the text box, QuickWatch displays the result on the first line of the spreadsheet. If you type an array, object, or structure variable, however, QuickWatch uses the spreadsheet to show additional detail. Plus sign (+) and minus sign (–) boxes appear. Click these boxes to expand or collapse your view of the variable.

If the variable is an object or a pointer to an object, QuickWatch automatically expands the variable to show the most important data at the top level. For example, suppose you had the following object:

```
int width = 186;
int height = 95;
Rectangle rect = new Rectangle(0, 0, width,height);
```

QuickWatch would display the following:

```
-rect = {...}
  java.lang.Object = {...}
  x = 0
  y = 0
  width = 186
  height = 95
```

If the variable is a reference to a Java object, QuickWatch automatically downcasts the reference. QuickWatch adds an extra member to the expanded object. This extra member, which looks like another base class, indicates the derived subclass. For example, if a variable declared as a reference to **Object** really references a **Rectangle object**, QuickWatch recognizes this fact and adds an extra member so that you can access the **Rectangle** object's members.

QuickWatch displays values in their default format. You can change the display format (to display Unicode characters, for example) using formatting symbols. For details, see Formatting Watch Variables.

{ewl msdncd, EWGraphic, jug20f 0 /a "build.bmp"} To view the value of a variable or expression using QuickWatch

- 1 Wait for the debugger to stop at a breakpoint.
- 2 From the Debug menu, choose QuickWatch.
The QuickWatch dialog box appears.
- 3 Type or paste the variable name or expression into the Expression text box.

- 4 Choose the Recalculate button.
- 5 Choose the Close button.

Tip The Expression drop-down list box contains the most recently used QuickWatch expressions.

{ewl msdncd, EWGraphic, jug20f 1 /a "build.bmp"} To quickly view the value of a variable using QuickWatch

- 1 When the debugger is stopped at a breakpoint, switch to a source window, and click the right mouse button on a variable (Var, for example).
- 2 From the pop-up menu, choose QuickWatch Var.
- 3 Choose the Recalculate button.
- 4 Choose the Close button.

When the program is paused at a breakpoint or between steps, you can change the value of any non-**final** variable in your program. This gives you the flexibility to try out changes and see their results in real time or to recover from certain logic errors.

{ewl msdncd, EWGraphic, jug20f 2 /a "build.bmp"} To modify the value of a variable using QuickWatch

- 1 From the Debug menu, choose QuickWatch.
- 2 In the Expression text box, type the variable name.
- 3 Choose the Recalculate button.
- 4 If the variable is an array or object, use the + box to expand the view until you see the value you want to modify.
- 5 Use the TAB key to move to the value you want to modify.
- 6 Type the new value, and then press ENTER.
- 7 Choose the Close button.

Tip To change the value of an object or array (including strings), modify the individual fields or elements. You cannot edit an entire array or object at once.

{ewl msdncd, EWGraphic, jug20f 3 /a "build.bmp"} To add a QuickWatch variable or expression to the Watch window

- 1 Use any of the procedures described previously to view the variable or expression in QuickWatch.
- 2 Choose the Add Watch button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Watch Window

Use the Watch window to specify variables and expressions that you want to watch while debugging your program. You can also modify the value of a variable using the Watch window.

The Watch window contains four tabs: Watch1, Watch2, Watch3, and Watch4—Each tab displays a user-specified list of variables and expressions in a spreadsheet field. You can group variables that you want to watch together onto the same tab. For example, you could put variables related to a specific window on one tab and variables related to a dialog box on another tab. You could watch the first tab when debugging the window and the second tab when debugging the dialog box.

{ewl msdncd, EWGraphic, jug21f 0 /a "build.bmp"} To add a variable or expression to the Watch window

- 1 From the View menu, choose Watch.
The Watch window appears.
- 2 Select a tab for the variable or expression.
- 3 Type, paste, or drag the variable name or expression into the Name column on the tab.
- 4 Press ENTER.
The Watch window evaluates the variable or expression immediately and displays the value or an error message.

If you add an array, or object variable to the Watch window, plus sign (+) or minus sign (–) boxes appear in the Name column. You can use these boxes to expand or collapse your view of the variable, as described in Spreadsheet Fields. For example, suppose you had the following object:

```
int width = 186;
int height = 95;
Rectangle rect = new Rectangle(0, 0, width,height);
```

The expanded variable in the Watch window displays the following:

```
-rect = {...}
  java.lang.Object = {...}
  x = 0
  y = 0
  width = 186
  height = 95
```

If the variable is a reference to a Java object, the Watch window automatically downcasts the reference. The Watch window adds an extra member to the expanded object. This extra member, which looks like another base class, indicates the derived subclass. For example, if a variable declared as a reference to type **Object** really references a **Rectangle object**, the Watch window recognizes this fact and adds an extra member so that you can access the **Rectangle** object's members.

The Watch window displays values in their default format. You can change the display format (to display Unicode characters, for example) using formatting symbols. For details, see Formatting Watch Variables.

Tip The Watch window does not display variable type information. You can view information for

a variable type by using the window's property page.

{ewl msdncd, EWGraphic, jug21f 1 /a "build.bmp"} To view type information for a variable

- 1 In the Watch window, select the line containing the variable whose type you want to see.
- 2 Click the right mouse button in the Watch window and choose Properties from the pop-up menu.
—or—
From the Edit menu, choose Properties.

{ewl msdncd, EWGraphic, jug21f 2 /a "build.bmp"} To remove a variable or expression from the Watch window

- 1 In the Watch window, select the line containing the variable or expression you want to remove.
- 2 Press the DEL key.

When the program is paused at a breakpoint or between steps, you can change the value of any non-**final** variable in your program. This gives you the flexibility to try out changes and see their results in real time, or to recover from certain logic errors.

{ewl msdncd, EWGraphic, jug21f 3 /a "build.bmp"} To modify the value of a variable using the Watch window

- 1 In the Watch window, double-click the value.
—or—
Use the TAB key to move the insertion point to the value you want to modify.
- 2 If the variable is an array, or object, use the + box to expand the view until you see the value you want to modify.
- 3 Type the new value, and press ENTER.

Tip To change the value of an object or array (including strings), modify the individual fields or elements. You cannot edit an entire array or object at once.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Formatting Watch Variables

You can change the display format of variables in the QuickWatch dialog box or in the Watch window using the formatting symbols in the following table.

Sy	Form	Valu	Disp
m	at	e	lays
bo			
I			
d,i	signe	0xF	-268
	d	000	3739
	decim	F06	15
	al	5	
	intege		
	r		
u	unsig	0x0	101
	ned	065	
	decim		
	al		
	intege		
	r		
o	unsig	0xF	0170
	ned	065	145
	octal		
	intege		
	r		
x,	Hexa	615	0x00
X	decim	41	00F0
	al	(dec	65
	intege	imal	
	r)	
l,h	long	004	0x0c
	or	060	22
	short	42,h	
	prefix	x	
	for: d,		
	i, u, o,		
	x, X		
f	signe	3./2.	1.50
	d		0000
	floati		
	ng-		
	point		
e	signe	3./2.	1.50
	d		0000
	scient		e+00
	ific		0
	notati		
	on		
g	signe	3./2.	1.5
	d		

[float](#)
[ing-point](#)
or

signed

scientific
notation,
which
ever
is
shorter

c	Single	0x00000065	"e"
	character		
s	String	0x00000012f0de8	"Hello world"
su	Unicode string		"Hello world"

To use a formatting symbol, type the variable name, followed by a comma and the appropriate symbol. For example, if **var** has a value of 0x0065, and you want to see the value in character form, type **var, c** in the Name column on the tab of the Watch window. When you press ENTER, the character-format value appears:

```
var, c = 'e'
```

To display a Unicode string in the Watch window or the QuickWatch dialog box, use the **su format** specifier. To display data bytes with Unicode characters in the Watch window or the QuickWatch dialog box, use the **mu format** specifier.

Note You can apply formatting symbols to arrays, references, and objects as unexpanded variables only. If you expand the variable, the specified formatting affects all members. You cannot apply formatting symbols to individual members.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Variables Window

The Variables window provides quick access to variables that are important in the program's current context. The window includes three tabs:

- The Auto tab displays variables used in the current statement and the previous statement.
- The Locals tab displays the variables that are local to the current method.
- The This tab displays the object referenced by **this**.

Each tab contains a spreadsheet with fields for the variable name and value. The debugger automatically fills in these fields.

You cannot add variables or expressions to the Variables window (use the Watch window for that), but you can expand or collapse the variables shown using the tree controls. You can expand an array or object variable in the Variables window if it has a plus sign (+) box in the Name field. If an array or object variable has a minus sign (–) box in the Name field, the variable is already fully expanded. To expand or collapse the variable, click the + or – box, as described in Spreadsheet Fields. For example, suppose you had the following object:

```
int width = 186;
int height = 95;
Rectangle rect = new Rectangle(0, 0, width,height);
```

The expanded variable in the Variables window would display the following:

```
-rect = {...}
  java.lang.Object = {...}
    x              = 0
    y              = 0
    width          = 186
    height         = 95
```

If the variable is a reference to a Java object, the Variables window automatically downcasts the reference. The Variables window adds an extra member to the expanded object. This extra member, which looks like another base class, indicates the derived subclass. For example, if a variable declared as a reference to a **Object** type really references a **Rectangle object**, the Variables window recognizes this fact and adds an extra member so that you can access the **Rectangle** object's members.

In addition to the tabs, the Variables window has a Context box on the toolbar that contains a copy of the current call stack in a drop-down list box. Use this list to specify the current scope of the variables displayed. The Content box is part of a toolbar, which you can hide using the right mouse button.

{ewl msdncd, EWGraphic, jug23f 0 /a "build.bmp"} To display the Variables window

- 1 From the View menu, choose Variables.
The Variables window appears.
- 2 Select the Auto tab, Locals tab, or This tab, according to the type of variables you want to see.

When the program is paused at a breakpoint or between steps, you can change the value of any non-**final** variable in your program. This gives you the flexibility to try out changes and see their results in real time or to recover from some logic error and continue.

The Variables window does not display variable type information. You can view type information for a variable by using the window's property page.

{ewl msdncd, EWGraphic, jug23f 1 /a "build.bmp"} To view type information for a variable

- 1 In the Variables window, select the Auto tab, the Locals tab, or the This tab.
- 2 Select the line containing the variable whose type you want to see.
- 3 Click the right mouse button in the Variables window, and choose Properties from the pop-up menu.

—or—

From the Edit menu, choose Properties.

Although you cannot delete variables from the Variables window, you can edit their values.

{ewl msdncd, EWGraphic, jug23f 2 /a "build.bmp"} To modify the value of a variable

- 1 In the Variables window, select the Auto tab, Locals tab, or This tab.
- 2 Select the line containing the variable whose type you want to modify.
- 3 If the variable is an array or object, use the + box to expand the view until you see the value you want to modify.
- 4 Double-click the value, or use the TAB key to move the insertion point to the value you want to modify.
- 5 Type the new value, and press ENTER.

Tip To change the value of an object or an array (including strings), modify the individual fields or elements. You cannot edit an entire array or object at once.

You can use the Variables window to examine method return values, as well as variables.

{ewl msdncd, EWGraphic, jug23f 3 /a "build.bmp"} To view the return value of a method

- 1 Step over or out of the method.
- 2 In the Variables window, select the Auto tab.
- 3 Click the Return Value icon, which appears in the Name column, as shown in Figure 12.1.
The method return value appears in the Auto tab. The Name column displays the return value as *Name* Returned, where *Name* is the name of the method.

Figure 12.1 Return Value Icon in Name Column

{ewc msdncd, EWGraphic, jug23f 4 /a "uguideFUN.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Limitations on the Variables Window

Some project settings and programming practices limit the ability of the Variables window to display variables:

- The Auto tab uses syntax coloring to determine which variables to display. If you turn off syntax coloring for a file, the Auto tab cannot display any variables for that file. To turn syntax coloring back on, select a source window, choose Properties from the Edit menu, and set Language to Java in the Source Windows property page.
- All three tabs use debugging information. If you build a section of code without debugging information, the tabs cannot display variables for that code.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Navigating From the Variables Window

You can navigate to a method's source code or disassembled bytecode from the Context box in the Variables window. This procedure displays the method's source code, if it is available, in a source window. If source code for the selected method is not available, it displays the method's bytecode in the Disassembly window.

{ewl msdncd, EWGraphic, jug25f 0 /a "build.bmp"} To navigate from the Variables window to a method's source or object code

- Select the method name from the Context drop-down list box in the Variables window toolbar.

This procedure changes the view of the program displayed in the Variables window and other debugger windows, but does not change the next line of execution or the value stored in the program counter.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Call Stack Window

During a debug session, the Call Stack window displays the stack of currently active method calls. When a method is called, it is pushed onto the stack. When the method returns, it is popped off the stack.

The Call Stack window displays the currently executing method at the top of the stack and older method calls below that. By default, the window also displays parameter types and values for each method call. You can display or hide parameter types and values using the Debug tab of the Options dialog box or the right mouse button pop-up menu.

{ewl msdncd, EWGraphic, jug26f 0 /a "build.bmp"} To display the Call Stack window

- From the View menu, choose Call Stack.

{ewl msdncd, EWGraphic, jug26f 1 /a "build.bmp"} To view the call stack for a method

- 1 Place the insertion point in the method.
- 2 From the Debug menu, choose Run To Cursor to execute your program to the location of the insertion point.

The Locals tab of the Variables window updates automatically to display the local variables for the method or procedure.

- 3 From the View menu, choose Call Stack.

The calls are listed in the calling order, with the current method (the most deeply nested) at the top.

Tip To run the program to the return address, select the method in the Call Stack window, and choose Run To Cursor from the Debug menu.

Tip To set or remove a breakpoint at a method return address, select the method in the Call Stack window, and choose the Insert/Remove Breakpoint toolbar button.

You can navigate to a method's source code or disassembled bytecode from the Call Stack window. This procedure displays the method's source code, if it is available, in a source window. If source code for the selected method is not available, it displays the method's bytecode in the Disassembly window.

{ewl msdncd, EWGraphic, jug26f 2 /a "build.bmp"} To navigate from the Call Stack window to a method's source or object code

- Double-click the method name in the Call Stack window.
—or—
Select the method name, and press ENTER.

This procedure changes the view of the program shown in the Variables window and other debugger windows, but does not change the next line of execution or the value stored in the program counter.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Controlling Call Stack Display

Note By default, the Call Stack window displays parameter values and types for each method. You can turn off the display of parameter values, types, or both using the Debug tab in the Options dialog box or by using the right mouse button pop-up menu. You cannot turn off the display of method names.

{ewl msdncd, EWGraphic, jug27f 0 /a "build.bmp"} To change the call stack display

- 1 From the Tools menu, choose Options.
- 2 Select the Debug tab.
- 3 Under Call stack window, select the check boxes for Parameter Values or Parameter Types, according to the information you want to display.

—or—

In the Call Stack window, click the right mouse button, and from the pop-up menu, choose Parameter Values or Parameter Types to toggle the display of that information.

Tip The Context box at the top of the Variables window contains a drop-down list of call stack methods. If you select one of these methods, the debugger window views change accordingly.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using the Disassembly Window

By default, the Disassembly window displays disassembled bytecode with source-code annotations and symbols. You can change these display options using the Options dialog box.

{ewl msdncd, EWGraphic, jug28f 0 /a "build.bmp"} To change the Disassembly window display options

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Debug tab.
- 3 Under Disassembly Window, select the appropriate check box for the display you want.
- 4 Choose OK.

The Disassembly window can be especially useful for debugging optimized code as well as source-code lines that contain multiple statements. Consider, for example, the following line of code:

```
x=1; y=7; z=3;
```

The source window treats each line of code as a unit. Using the source window, you cannot step from one statement on a source-code line to the next, or set a breakpoint on any statement other than the first.

The Disassembly window operates on bytecode instructions instead of source-code statements or lines. Using the Disassembly window, you can set a breakpoint on any instruction. If you use the Step Into or Step Over command while the Disassembly window has focus, the debugger steps through your program instruction-by-instruction instead of line-by-line. Viewing and stepping through your code by bytecode instructions can be especially useful when you are debugging optimized code.

Using the Disassembly window, you can display the bytecode created for the source code being debugged.

{ewl msdncd, EWGraphic, jug28f 1 /a "build.bmp"} To display the Disassembly window

- From the View menu, choose Disassembly.

{ewl msdncd, EWGraphic, jug28f 2 /a "build.bmp"} To switch between corresponding locations in the source and Disassembly windows

- In the source window, click the right mouse button, and choose Go To Disassembly from the pop-up menu.
—or—
In the Disassembly window, click the right mouse button, and choose Go To Source from the pop-up menu.

{ewl msdncd, EWGraphic, jug28f 3 /a "build.bmp"} To view disassembly code at a specified location using the Go To Disassembly command

- In the source window, select the line of code that you want to see the disassembled bytecode for.
- Click the right mouse button.
A pop-up menu appears.
- Select Go To Disassembly from the pop-up menu.

The Disassembly window appears at the location of your selection.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Debugging Methods and Strategies

Debugging Compiler Errors offers some suggestions for debugging problems that stop you from building.

The How Can I...? topic offers some suggestions for assorted situations you may encounter while debugging.

Suggestions for debugging specific types of code are found in:

- Debugging Exceptions
- Debugging Threads

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Debugging Compiler Errors

The first step in debugging is to fix language syntax errors. The Output window displays errors that prevent a program from being built and provides the filename, line number, and error number. The Output window behaves like a source window; you can copy and print information from the window. If the status bar is displayed, it gives a summary of the current error.

If you don't understand an error message, move the insertion point to the error number, and press the F1 key (in the default keyboard mapping) to display online information about it.

{ewl msdncd, EWGraphic, jug30f 0 /a "build.bmp"} To move through the list of errors

- In the Output window, double-click the error, or select the error and press ENTER.
—or—
Click the right mouse button in the Output window, and choose Go To Error/Tag from the pop-up menu.
—or—
Press F4 (in the default keyboard mapping) to select the next error.
—or—
Press SHIFT+F4 (in the default keyboard mapping) to select the preceding error.
As each error is selected in the Output window, the corresponding line containing the error is selected in the source window.

You can move to any line number in a source file.

{ewl msdncd, EWGraphic, jug30f 1 /a "build.bmp"} To move to a specific line

- 1 From the Edit menu, choose Go To.
The Go To dialog box appears.
- 2 In the Line Number box, type a line number.
- 3 Choose the Go To button.

If you type a line number greater than the last line in your source file, the editor moves to the end of the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

How Can I...?

This section provides suggestions on how to handle some common, and some not-so-common, debugging situations.

{ewl msdncd, EWGraphic, jug31f 0 /a "build.bmp"} How can I perform source-level debugging in the standard Java class libraries, like `java.lang` or `java.awt`?

- You need the source code for the Java class libraries. You can install the source code by running the Visual J++ Setup program and choosing a Custom installation. Select "Java Class Library Source Code," clear the check boxes for everything else, and then continue. This extracts the .JAVA files from the CLASSES.ZIP file included with Internet Explorer.

You can also extract the .JAVA files manually by running the JavaSrc tool, which is in the same directory as your CLASSES.ZIP file. From the command line, enter "javasrc classes.zip" to extract the .JAVA files.

{ewl msdncd, EWGraphic, jug31f 1 /a "build.bmp"} I set a breakpoint at a line in my source code, but I'm actively editing the code as I debug. When I rebuild the project, I get an error message telling me that the breakpoint has moved. How can I stop this from happening?

- If possible, set the breakpoint at the beginning of the method, by specifying the method name, instead of setting the breakpoint on a line number. Breakpoints on source-code lines stay on the same line number. If you edit the code, changing the number of lines, the breakpoint may no longer be on a line with a valid statement. Breakpoints set at the beginning of the method remain with the method, regardless of what source-code line the method begins on. For more information, see Quick Methods for Location Breakpoints.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Debugging Exceptions

The exception-handling facility in Java allows programs to handle abnormal and unexpected situations in an orderly, structured manner. When a method detects an exception that must be handled, it notifies the handler using throw. The exception handler receives the notification using catch. If no catch handler exists for an exception, the program typically causes the Java interpreter to print an error message and print a stack trace and exit. If you are debugging a program in Visual J++, however, the debugger notifies you that the exception was not caught.

When you are debugging in Visual J++, you can use the Exceptions dialog box to specify how the debugger is to handle each specific type of exception. In this dialog box, you can set one of two options—Stop Always or Stop If Not Handled—for each exception type that can occur in your program.

If you select Stop If Not Handled for an exception, the debugger writes a message to the Output window when an exception occurs, but does not halt the program and notify you with a dialog box unless the exception handler fails to handle the exception. At that point, it is too late to fix the problem or examine the source code to see where the exception occurred. (The program is already past the point where the exception occurred and is executing in the exception handler.)

If you select Stop Always for an exception, the debugger stops the program and notifies you immediately when an exception occurs, before any handler code is invoked. When this happens, you can look at the source window to see where the exception occurred. You can use the Watch and Variables windows, as well as QuickWatch, to see current variable contents. In some cases, you can fix the exception yourself by modifying the variable contents. When you continue the program after the exception, a dialog box appears asking if you want to pass the exception back to the program's exception handlers. If you fixed the problem, choose the No button. Otherwise, choose the Yes button to invoke the exception handler. If the exception handler cannot fix the problem, the debugger halts the program and notifies you again, just as if you had selected Stop If Not Handled.

The Exceptions list box in the Exceptions dialog box contains Java Exception—currently the only system exception available. You can remove the Java Exception or add exceptions of your own. This information is saved in the *project.MDP* file, which persists with the project. If an exception is not included in this list, the debugger treats it as a Stop If Not Handled exception.

Note You can stop on all Java exceptions when they are thrown by choosing Java Exception in the Exceptions dialog box, and choosing Stop Always. Choosing Stop If Not Handled will cause the debugger to stop on Java exceptions which are not caught.

{ewl msdncd, EWGraphic, jug32f 0 /a "build.bmp"} To add a new exception to the Exceptions list box

- 1 From the Debug menu, choose Exceptions.
The Exceptions dialog box appears.
- 2 In the Number box, type the exception number for the user-defined exception.
- 3 Optionally, type the name of the user-defined exception in the Name box.
- 4 Optionally, under Action, select the Stop Always or Stop If Not Handled option button.

- 5 Choose the Add button.
- 6 Choose OK.

You can change any parameter associated with an exception.

{ewl msdncd, EWGraphic, jug32f 1 /a "build.bmp"} To change an exception parameter

- 1 From the Debug menu, choose Exceptions.
The Exceptions dialog box appears.
- 2 In the Exceptions list box, select the exception.
- 3 Change any parameter, such as the name or the action.
- 4 Choose the Change button.
- 5 Choose OK.

You can remove any exception from the Exceptions list box.

{ewl msdncd, EWGraphic, jug32f 2 /a "build.bmp"} To remove an exception from the Exceptions list box

- 1 From the Debug menu, choose Exceptions.
The Exceptions dialog box appears.
- 2 In the Exceptions list box, select the exception.
- 3 Choose the Remove button.
When you delete an exception from the Exceptions list box, its action reverts to Stop If Not Handled.
- 4 Choose OK.

If you wish to restore system exceptions to the list, choose the Reset button.

{ewl msdncd, EWGraphic, jug32f 3 /a "build.bmp"} To restore all default system exceptions to the Exceptions list

- 1 From the Debug menu, choose Exceptions.
The Exceptions dialog box appears.
- 2 Choose the Reset button.
All default system exceptions are restored to the Exceptions list box without disturbing any of the user-defined exceptions that have been added.
- 3 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Debugging Threads

You can use the Microsoft Developer Studio debugger to debug multithreaded applications.

A thread is a path of execution within a process. A process is an executing instance of a program. Launching Notepad, for example, starts a process that has a single thread. The startup code passes this primary thread to the operating system in the form of a method address (for example, **main()**). When the primary thread terminates, so does the process.

You can create additional threads in your application code. These threads can handle background or maintenance tasks that proceed without the user's attention.

When debugging a multithreaded program, you can select a single thread using the Threads dialog box.

{ewl msdncd, EWGraphic, jug33f 0 /a "build.bmp"} To display the Threads dialog box

- From the Debug menu, choose Threads.

The Threads dialog box displays a list of all threads that exist in the application. Using this list, you can set focus on, suspend, or resume a thread.

{ewl msdncd, EWGraphic, jug33f 1 /a "build.bmp"} To set focus on a thread

- 1 In the Threads dialog box, select a thread from the Thread list.
- 2 Choose the Set Focus button.

{ewl msdncd, EWGraphic, jug33f 2 /a "build.bmp"} To suspend a thread

- 1 In the Threads dialog box, select a thread from the Thread list.
- 2 Choose the Suspend button.

{ewl msdncd, EWGraphic, jug33f 3 /a "build.bmp"} To resume execution of a thread

- 1 In the Threads dialog box, select a thread from the Thread list.
- 2 Choose the Resume button.

The Thread list in the Threads dialog box displays status information on each thread as follows:

- The Thread ID column contains the DWORD that uniquely identifies each thread. When you set focus on a thread, an asterisk (*) appears next to its thread ID.
- The Suspend column contains the suspension number of each thread. This number, which can vary from 0 through 127, is incremented each time you suspend the thread and decremented each time you resume the thread.
- The Priority column contains the thread priority. A Java thread's priority is always Normal.
- The Location column contains the method name or address associated with the thread. You can choose to see either the method name or address.

{ewl msdncd, EWGraphic, jug33f 4 /a "build.bmp"} To view the method name associated with each thread

- In the Threads dialog box, select Name.

{ewl msdncd, EWGraphic, jug33f 5 /a "build.bmp"} To view the address associated with each thread

- In the Threads dialog box, select Address.

If Name is selected, the current method name is displayed if it is known by the debugger. If no method is known, the address is displayed. If Address is selected, the current address is displayed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Compiler Options

This section describes the compiler options available from the Java tab of the Project Settings dialog box: Warning Levels, Generate Debug Info, and Reset.

The Java tab of the Project Settings dialog is shown in Figure 13.1.

{ewl msdncd, EWGraphic, jug0o 0 /a "build.bmp"} To display the Project Settings dialog box

- From the Build menu, choose Settings.

For information about the options on the Java tab, choose an item from the following list:

- Warning Levels
- Optimization Options
- Generate Debug Information
- Reset

Note All compiler command-line options may be entered into the Project Options text box at the bottom of the Java tab. For more information about compiler options not displayed on the Java tab, see JVC Reference.

Figure 13.1 The Java Tab

{ewc msdncd, EWGraphic, jug0o 1 /a "uguideJAVTB.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Warning Levels

The following options control the number of warning messages produced by the compiler:

None Turns off all warning messages. Command-line equivalent: /w0

Level 1 Displays only severe warnings. Command-line equivalent: /w1

Level 2 Displays less severe warnings, such as the use of methods with no declared return type, failure to put return statements in methods that are not void, and data conversions that would cause loss of data or precision. This is the default warning level setting. Command-line equivalent: /w2

Level 3 Displays lesser severe warnings. Command-line equivalent: /w3

Level 4 Displays least severe warnings. Command-line equivalent: /w4

Compiler error messages begin with J0; warning messages begin with J5. Online help (F1) describes the errors and warnings, and indicates each warning level and potential problems (rather than actual coding errors) for statements that may not compile as you intend.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Optimizations Options

The Full Optimization check box (Figure 13.2) determines how the compiler fine tunes the performance of your program. The options on the Java tab allow you to request either full or default optimization for the selected project or projects.

Figure 13.2 Compiler Settings on the Java Tab

```
{ewc msdncd, EWGraphic, jug2o 0 /a "uguideJAVTB.BMP"}
```

You can select one of the following optimization categories:

Disable Optimization Turns off all optimization of your code except for the default setting. Disabling optimization simplifies debugging because it suppresses code movement. Command-line equivalent: /O-

Note The default optimization setting for all Visual J++ projects is /O:J, optimize bycode jumps. To disable the default optimization, type /O:J- in the Project Options text box on the Java tab.

Full Optimization Creates the fastest code in the majority of cases.

The effect of using this option is the same as specifying the following options on the command line: /O:J /O:I

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Generate Debug Information

This option produces an interpreted .CLASS file containing information that is used by the debugger. Command-line equivalent: /g

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Reset

The Reset button resets the project settings of a project or a file to the settings that existed when the project or file was created. This button is available if both of the following conditions are met:

- At least one file or project is selected in the Settings For pane of the Project Settings dialog box.
- The settings of the selection have changed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Customizing Microsoft Developer Studio

You can customize Microsoft Developer Studio by:

- Arranging the layout of windows and toolbars.
- Adding your favorite toolbar buttons to the toolbar.
- Assigning shortcut keys to commands.
- Adding your tools to the Tools menu.
- Specifying directories for build utilities, include files, and libraries.

You can also arrange the display area in a way that best suits your development needs. Some arrangements are maintained with each project. For instance, you can size editor windows, move them to convenient locations, and automatically save these locations with your project.

Other arrangements are maintained globally. For instance, you can display some windows in one layout while you are editing your files or building your project, and use another layout when you are debugging.

Some windows can either be fixed along the application window border or moved anywhere on your screen. These windows are called docking tool windows.

Some Developer Studio commands are assigned default shortcut keys. In Developer Studio, you can:

- Delete existing assignments for shortcut keys.
- Replace default shortcut keys with different ones.
- Assign shortcut keys to commands that have none by default.
- Assign multiple shortcut keys to a command.

Use these assignments to set your own shortcut keys—ones that are familiar and natural for you to use, or that have some easily remembered value.

Developer Studio has F1 source-file help for language keywords and method calls. In some cases, language elements may have entries in multiple topics or information titles (an information title is a .MVB file such as Books Online or Application Help). If you press F1 on a source-file keyword, say `getAppletInfo`, Developer Studio looks for information on the keyword in the current information title. If multiple topics exist for the keyword, Developer Studio displays the Select Reference dialog box. The Select Reference dialog box lists not only the topics but also other information titles with topics for the keyword. You can select another information title if you wish. For further information, see Finding Information.

Note You can also customize other aspects of Developer Studio. To find out about customizing text editor windows and their use of fonts or colors, see Using the Text Editor. To find out about customizing settings for debugging, see Using the Debugger.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Working with Window Types

Microsoft Developer Studio windows are shown in the following table.

Type	Attributes	Layout associated with
Document windows	Position and size can be changed only within the application window. Can be maximized and minimized.	Project
Docking tool windows and tool bars	Attach to <i>docks</i> along the borders of the application window, or float anywhere on your screen. A toolbar is a type of docking tool window. All docking	Editing or debugging

g tool
windo
ws
except
toolbar
s can
conver
t to
[docu
ment
windo
ws.](#)

[Full-
scre
en
mod
e](#) Editing Editing
windo or
w debug
expan ging
ds to
the
size of
the
entire
screen

The layout for window types—that is, their visibility, position, and size—is associated either with a project, or with editing or debugging operations. Once you have chosen a layout, that layout is **persistent**. If you close a project and later open it again, the **document windows** have the last layout that you used: the same windows are open, and have the same sizes and positions. When you create layouts of docking tool windows or toolbars, either for editing, debugging, or full-screen mode, those layouts are used for all subsequent sessions until you change them again.

All window types can display context pop-up menus with commands appropriate for the window's current state. For example, the context menu for an editor window (a type of **document** window) displays the Cut, Copy, and Paste commands while editing, but displays the Insert/Remove Breakpoint and QuickWatch commands while debugging. Click the right mouse button in the window to display the pop-up menu.

There is also a pop-up menu associated with the dock along the border of the application window. If you click the right mouse button in the dock area or on a **toolbar**, the menu displays commands to show or hide the docking tool windows, and to customize the toolbars. The various pop-up menus are shown in Figure 14.1.

Figure 14.1 Pop-up Menus Displayed with the Right Mouse Button

{ewc msdncd, EWGraphic, jug1g 0 /a "uguideSHRT.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Document Windows

The following windows are document windows:

- Editor windows
- Resource Template window

Document windows are associated with the project workspace. Developer Studio records their positions, sizes, any selections made in them when you close a project. When you open the project again, Developer Studio restores these characteristics.

Document windows can also display pop-up menus with commands appropriate for the window's current state. Click the right mouse button in the document window to display the pop-up menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Positioning Document Windows

You can position the document windows for a project to suit your preferences. These positions are retained when you close the project. When you open the project again, Developer Studio restores these window positions, opens the necessary files, and displays their contents in the windows, with selections that you have made.

{ewl msdncd, EWGraphic, jug3g 0 /a "build.bmp"} To display the pop-up menu for a document window

- Click the right mouse button in the window.

{ewl msdncd, EWGraphic, jug3g 1 /a "build.bmp"} To move a document window

- From the window's pop-up menu, select Docking View. Point to the title bar, and drag the window to the location you want.

{ewl msdncd, EWGraphic, jug3g 2 /a "build.bmp"} To size a document window

- Point to the window border and drag it to the size you want.

{ewl msdncd, EWGraphic, jug3g 3 /a "build.bmp"} To tile document windows

- From the Window menu, choose Tile Horizontally.
—or—
From the Window menu, choose Tile Vertically.

{ewl msdncd, EWGraphic, jug3g 4 /a "build.bmp"} To overlap document windows

- From the Window menu, choose Cascade.

{ewl msdncd, EWGraphic, jug3g 5 /a "build.bmp"} To split document windows

- 1 From the Window menu, choose Split.
- 2 Drag the splitter bars in the window to the location you want, and click the mouse to set the location of the splitter bars.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Selecting Document Windows to Display When Opening a Project

You can specify whether to display project documents upon opening a project.

{ewl msdncd, EWGraphic, jug4g 0 /a "build.bmp"} To display project documents upon opening a project

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Workspace tab.
- 3 Select the check box for Reload Documents When Opening Project.
- 4 Choose OK .

{ewl msdncd, EWGraphic, jug4g 1 /a "build.bmp"} To not display project documents upon opening a project

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Workspace tab.
- 3 Clear the check box for Reload Documents When Opening Project.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Docking Tool Windows

With docking tool windows, you can customize the workspace by:

- Showing or hiding the docking tool window.
- Changing the display mode from docked to floating.
- Resizing any floating docking tool window.
- Giving docking tool windows the display characteristics of a document.

Docking tool windows are available only when editing or debugging, even if you have selected the characteristics of document windows. The debugging docking tool windows, such as the Watch, QuickWatch, and Variable windows, are available only during the debug process.

The following windows are docking tool windows:

- Output window
- Watch window
- Variables window
- Call Stack window
- Disassembly window
- Project Workspace window
- InfoViewer Topic window

Tip A docking tool window can display pop-up menus with commands appropriate for the window's current state. To display the window's pop-up menu, click the right mouse button.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Showing and Hiding Docking Tool Windows

You can show or hide the Output, Project Workspace, and InfoViewer Topic windows at any time. The debugging windows are available only during the debug process.

{ewl msdncd, EWGraphic, jug6g 0 /a "build.bmp"} To show a docking tool window

- 1 From the View menu, choose Toolbars.

The Toolbars dialog box appears.

- 2 Select the docking tool window that you want to show.

—or—

With the right mouse button, click the border of a docking tool window (see Working with Window Types).

The pop-up menu appears. A check mark next to a window name indicates that the window is currently displayed.

- 3 Select the unchecked docking tool window that you want to show.

The window either appears in its default location or in the last location that you assigned it.

{ewl msdncd, EWGraphic, jug6g 1 /a "build.bmp"} To hide a docking tool window

- 1 Click in the window to make it active.

- 2 From the Window menu, choose Hide.

—or—

With the right mouse button, click inside the window, and from the pop-up menu, choose Hide.

If the docking tool window is currently displayed as a document window, double-click the control box in the upper-left corner of the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Positioning Docking Tool Windows

The positions of a docking tool window are not associated with the current project; the positions remain the same no matter which project you open. However, the positions can be different for either editing or debugging. You can create one layout of docking tool windows for editing, and another for debugging. When you switch from editing to debugging, the layout automatically changes.

Docking tool windows can have either of two display modes: floating or docked.

Floating Mode

In floating mode, a docking tool window has a thin title bar and can appear anywhere on your screen. A floating window is always on top of all other windows.

Figure 14.2 Floating Variables Window

```
{ewc msdncd, EWGraphic, jug7g 0 /a "uguideFLTW.BMP"}
```

Docked Mode

In docked mode, a docking tool window is fixed to a dock along any of the four borders of the main Microsoft Developer Studio window.

Figure 14.3 Docked Variables Window

```
{ewc msdncd, EWGraphic, jug7g 1 /a "uguideDOC2.BMP"}
```

You can specify whether tool windows appear as docked windows or as floating windows.

{ewl msdncd, EWGraphic, jug7g 2 /a "build.bmp"} To change a docked window to a floating window

- 1 Point to a blank area on the window border.
 - 2 Drag the window away from the dock to the position that you want.
- or—
Double-click in the window border.

{ewl msdncd, EWGraphic, jug7g 3 /a "build.bmp"} To dock a floating window

- 1 Point to the title bar of the docking tool window.
 - 2 Drag the window to any of the four borders of the application window.
- or—
Double-click the window title bar to return the window to its previous docked location.

A docking tool window stretches to fill the entire border to which you drag it, as shown in Figure 14.4. A toolbar changes to a single row or column and takes the space required by its tool buttons.

Figure 14.4 Window in Floating and Docked Modes

```
{ewc msdncd, EWGraphic, jug7g 4 /a "uguideDOCK.BMP"}
```

{ewl msdncd, EWGraphic, jug7g 5 /a "build.bmp"} To position a floating window over a dock

- 1 Point to the title bar of the window.
- 2 Hold down the CTRL key, and drag the window over any dock area of the application window.
The window moves into position over the dock, but remains a floating window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Sizing Docking Tool Windows

You can resize any floating window in any direction. You can also size docked windows by moving their splitter bar or bars. If two or more tool windows are in the same dock, you can size them by moving the splitter bar between them.

{ewl msdncd, EWGraphic, jug10g 0 /a "build.bmp"} To resize a docking tool window

- 1 Point to the border of a docked or floating window.
The mouse pointer turns into a sizing arrow.
- 2 Drag the splitter bar or border to resize the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Changing Docking Tool Window Characteristics

You can use the Options dialog box on the Tools menu to give docking tool windows the display characteristics of document windows.

Note Toolbars cannot have document window characteristics, even though toolbars are docking tool windows.

A docking tool window can appear as a docked window, a floating window, or a document window. Although a docking tool window can appear with any of these characteristics, it remains a docking tool window.

{ewl msdncd, EWGraphic, jug11g 0 /a "build.bmp"} To enable or disable the document characteristics of a docking tool window

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Workspace tab.
- 3 In the Docking Views list box, select the check box for a window to enable its docking characteristics. Clear the check box for a window to disable its docking characteristics.
An unselected window behaves like a document window.

Note By default, the Disassembly and InfoViewer Topic windows have the characteristics of document windows.

- 4 Choose OK.

Alternatively, clear the check box for docking tool windows that you want to have the characteristics of document windows.

{ewl msdncd, EWGraphic, jug11g 1 /a "build.bmp"} To quickly switch between docking and document characteristics in a docking tool window

- 1 Click the right mouse button inside the window.
- 2 From the pop-up menu, choose Docking View.

If the docking tool window has document characteristics, it changes to a docked window. If the docking tool window has docking characteristics, it changes to a document window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Working with Toolbars

Toolbar buttons can correspond to Microsoft Developer Studio menu commands. These buttons provide a convenient way of carrying out frequently used commands.

When Developer Studio starts up in its standard configuration after installation, it displays the Standard toolbar and command choices. You can use these standard choices, or you can choose to display other toolbars and even choose which command buttons to display on those toolbars.

Note The toolbar categories that appear initially depend on the packages that you have installed on your system.

Because toolbars are docking tool windows, you can either fix a toolbar along a border of the application window, or turn it into a floating window that can move anywhere on your screen.

In addition to the Standard toolbar, Developer Studio displays toolbars that reflect the currently open editors. For example, if you open the resource file and then open a bitmap resource, Developer Studio displays the toolbars associated with the Image editor. Also, the current state of the program determines whether the tools on any given toolbar are enabled or disabled.

Depending on which editors are open and whether you are editing, debugging, or using full-screen mode, toolbar positions are not associated with the current project. For example, you can create one layout for editing, another layout for debugging, and another layout for full-screen mode. When you switch among editing, debugging, and full-screen mode, the layout automatically changes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Showing and Hiding Toolbars

When Microsoft Developer Studio starts up in its standard configuration after installation, it displays the predefined toolbars on its dock. The toolbar categories that appear depend on the software installed on your system. However, you can choose which toolbars you want to show at any time.

{ewl msdncd, EWGraphic, jug13g 0 /a "build.bmp"} To show or hide a toolbar using the main menu

- 1 From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2 Select the check boxes for the toolbars that you want to show.
- 3 Clear the check boxes for the toolbars that you want to hide.
Each selected toolbar appears immediately, in its default location, or in the last location that you assigned to it. Each hidden toolbar disappears immediately.
- 4 Choose Close.

{ewl msdncd, EWGraphic, jug13g 1 /a "build.bmp"} To show or hide a toolbar using the pop-up menu

- 1 Click the right mouse button on a toolbar, either floating or docked.
The pop-up menu appears. A check mark next to a toolbar name indicates that the window is currently displayed.
- 2 Select the unchecked toolbars that you want to show.
- 3 Select the checked toolbars that you want to hide.
Each selected toolbar appears immediately. It appears in its default location, or in the last location that you assigned it. Each hidden toolbar disappears immediately.
- 4 Choose Close.

{ewl msdncd, EWGraphic, jug13g 2 /a "build.bmp"} To hide a floating toolbar

- Click the close box in the upper-right corner of the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Showing ToolTips

You can display the name of a tool when you place the cursor on the tool button. You can also display the shortcut keys associated with the tool button.

{ewl msdncd, EWGraphic, jug14g 0 /a "build.bmp"} To show ToolTips

- 1** From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2** Select Show ToolTips if you want to display the name of a tool when you place the cursor on the toolbar button.
- 3** Select With Shortcut Keys if you want to display in the tool tip the shortcut keys associated with the tool button.
- 4** Choose Close.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a Custom Toolbar

You can create a custom toolbar and add any tool button to it. You can either give the new toolbar a title, or use the default title created by Microsoft Developer Studio.

{ewl msdncd, EWGraphic, jug15g 0 /a "build.bmp"} To create a named toolbar

- 1 From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2 Choose New.
The New Toolbar dialog box appears.
- 3 In the Toolbar Name text box, type the name of your custom toolbar.
- 4 Choose OK.
Two windows appear:
 - In the upper-left corner of your application window, a new toolbar window with the name that you specified appears.
 - The Customize dialog box appears.
- 5 In the Customize dialog box, select the Toolbars tab.
The categories list box in this tab shows categories of buttons. When you select a category, the buttons frame displays all the buttons in that category. Each button represents a command.
- 6 In the Categories list box, select a category.
- 7 Drag the buttons that you want from the selected category onto the custom toolbar.
- 8 Repeat steps 6 and 7 until you have all the buttons that you want on your toolbar.
- 9 Choose Close.

{ewl msdncd, EWGraphic, jug15g 1 /a "build.bmp"} To quickly create a toolbar with a default name

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 In the Customize dialog box, select the Toolbars tab.
The categories list box in this tab shows categories of buttons. When you select a category, the buttons frame displays all the buttons in that category. Each button represents a command.
- 3 In the Categories list box, select a category.
- 4 Drag the first button from the selected category onto any area of your screen (except an existing toolbar).
The first button creates a toolbar named Toolbar n , where n is 1, 2, 3, 4, and so on.
- 5 Continue to select a category and drag buttons onto the new toolbar until you have all the buttons you want on your toolbar.
- 6 Choose Close.

Note A toolbar with only a few buttons may be too short to fully display its name.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Modifying a Toolbar

You can easily add, remove, arrange, or copy toolbar buttons. You can even rename a custom toolbar.

{ewl msdncd, EWGraphic, jug16g 0 /a "build.bmp"} To add a button to a toolbar

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 In the Customize dialog box, select the Toolbars tab.
The categories list box in this tab shows categories of buttons. When you select a category, the buttons frame displays all the buttons in that category. Each button represents a command.
- 3 In the Categories list box, select a category.
- 4 Drag a button from the Buttons frame onto the toolbar.
- 5 Repeat steps 3 and 4 until you have all the buttons you want on your toolbar.
- 6 Choose Close.

{ewl msdncd, EWGraphic, jug16g 1 /a "build.bmp"} To remove a button from a toolbar

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Toolbars tab.
- 3 Drag away from the toolbar the button that you want to remove.
- 4 Choose Close.

{ewl msdncd, EWGraphic, jug16g 2 /a "build.bmp"} To move a button on a toolbar

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Toolbars tab.
- 3 Drag the button to a new location on the same toolbar or on another displayed toolbar.
- 4 Choose Close.

{ewl msdncd, EWGraphic, jug16g 3 /a "build.bmp"} To quickly move a button on a displayed toolbar

- Hold down the ALT key, and drag the button to a new location on the same toolbar or on another displayed toolbar.

{ewl msdncd, EWGraphic, jug16g 4 /a "build.bmp"} To copy a button from a toolbar

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Toolbars tab.
- 3 Hold down the CTRL key, and drag the button from the Buttons frame to its new location on the same toolbar or on another displayed toolbar.

4 Choose Close.

{ewl msdncd, EWGraphic, jug16g 5 /a "build.bmp"} To quickly copy a button on a toolbar

- Hold down the ALT+CTRL key combination, and drag the button from another toolbar to a new location on the same toolbar or on another displayed toolbar.

Note If you hold down the CTRL key, or a CTRL key combination, and drag a button onto an area where there is no existing toolbar, Microsoft Developer Studio creates a new toolbar with a default name.

{ewl msdncd, EWGraphic, jug16g 6 /a "build.bmp"} To insert a space between buttons on a toolbar

1 From the Tools menu, choose Customize.

The Customize dialog box appears.

2 Select the Toolbars tab.

3 Perform one or both of the following actions:

- To insert a space before a button that is either not followed by a space, or is followed by a space that you want to close, drag the button to the right or down until it overlaps the next button about halfway.
- To insert a space before a button that is followed by a space that you want to retain, drag the button across the existing space until the button just touches or overlaps the next button.

4 Choose Close.

{ewl msdncd, EWGraphic, jug16g 7 /a "build.bmp"} To close up a space between buttons on a toolbar

1 From the Tools menu, choose Customize.

The Customize dialog box appears.

2 Select the Toolbars tab.

3 Drag one of the buttons across the space toward the other button until the buttons overlap about halfway.

Dragging a button more than halfway past an adjacent button not only fails to close the space between them, but also inserts a new space (if one does not already exist) on the opposite side of the button being dragged.

4 Choose Close.

{ewl msdncd, EWGraphic, jug16g 8 /a "build.bmp"} To resize a combo box on a toolbar

1 From the Tools menu, choose Customize.

The Customize dialog box appears.

2 Select the Toolbars tab.

3 On the toolbar, select the combo box that you want to resize.

4 Drag the right edge of the combo box to the size that you want.

5 Choose Close.

{ewl msdncd, EWGraphic, jug16g 9 /a "build.bmp"} To rename a custom toolbar

- 1** From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2** In the Toolbars list, select the toolbar that you want to rename.
- 3** In the Toolbar Name text box, type the new name for the toolbar.
- 4** Choose Close.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Resetting a Toolbar

If you have modified a predefined toolbar, either by adding or removing buttons, you can easily restore its default settings.

{ewl msdncd, EWGraphic, jug17g 0 /a "build.bmp"} To reset a toolbar

- 1** From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2** In the Toolbars list box, select the toolbar that you want to reset.
- 3** Choose Reset.
- 4** Choose Close.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Deleting a Toolbar

You can delete any custom toolbar that you have created.

{ewl msdncd, EWGraphic, jug18g 0 /a "build.bmp"} To delete a custom toolbar

- 1** From the View menu, choose Toolbars.
The Toolbars dialog box appears.
- 2** In the Toolbars list box, select the toolbar that you want to delete.
You cannot delete any of the predefined toolbars.
- 3** Choose Delete.
- 4** Choose Close.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Docking Toolbars

When Developer Studio starts up in its standard configuration after installation, it displays the Standard toolbar on the top dock of the main application window, as shown in Figure 14.5.

Figure 14.5 Standard Toolbar Layout

```
{ewc msdncd, EWGraphic, jug19g 0 /a "uguideFLTT2.BMP"}
```

Toolbars can have either of two display modes: floating or docked.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Floating Mode

In floating mode, a toolbar has a thin title bar and can appear anywhere on your screen. A floating toolbar is always on top of all other windows. You can modify the size or position of a toolbar when it is floating.

Figure 14.6 Floating Toolbar

```
{ewc msdncd, EWGraphic, jug20g 0 /a "uguideFLTT2.BMP"}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Docked Mode

In docked mode, a toolbar is fixed to a dock along any of the four borders of the application window. You cannot modify the size of a toolbar when it is docked.

Figure 14.7 Docked Toolbar

```
{ewc msdncd, EWGraphic, jug21g 0 /a "uguideDOCT.BMP"}
```

You can dock any toolbar regardless of whether it is predefined by Microsoft Developer Studio or is a custom one created by you. Moving a toolbar from its docked position automatically converts it into a floating toolbar.

{ewl msdncd, EWGraphic, jug21g 1 /a "build.bmp"} To change a docked toolbar to a floating toolbar

- 1 Point to a blank area in the toolbar.
- 2 Drag the toolbar away from the dock to the position that you want.

{ewl msdncd, EWGraphic, jug21g 2 /a "build.bmp"} To dock a floating toolbar

- 1 Point to the toolbar title bar or a blank area in the toolbar.
- 2 Drag the toolbar to any of the four borders of the application window.

As the mouse pointer reaches the border, the toolbar window assumes a shape appropriate to the docking location. Along the top and bottom borders, it becomes a single horizontal row of buttons; along the sides, it becomes a single vertical row.

{ewl msdncd, EWGraphic, jug21g 3 /a "build.bmp"} To quickly move a toolbar onto or off of the toolbar dock

- Double-click a blank area in a docked toolbar, or the title bar of a floating toolbar.
If you double-click a docked toolbar, it moves to its previous floating position.
If you double-click the title bar of a floating toolbar, it moves to its last toolbar dock. If the toolbar has not been docked before, it moves to a new row in the toolbar dock below the menu bar.

{ewl msdncd, EWGraphic, jug21g 4 /a "build.bmp"} To position a floating toolbar over a dock

- 1 Point to a blank area of the toolbar or its title bar.
- 2 Hold down the CTRL key, and drag the toolbar over any dock area of the application window.
The toolbar moves into position over the dock, but remains a floating toolbar.

Tip The orientation of a docked toolbar generally corresponds to the orientation of the dock. Toolbars dock vertically on vertical docks, and horizontally on horizontal docks. To switch a toolbar's docked orientation between horizontal and vertical, press or release the SHIFT key as you drag and drop the toolbar.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Sizing Floating Toolbars

You can resize any floating toolbar. The toolbar changes the row and column arrangements of its buttons to accommodate whatever new orientation you give it. The toolbar takes the least amount of space necessary to display all of its buttons.

Note You cannot change the size or orientation of a docked toolbar.

{ewl msdncd, EWGraphic, jug22g 0 /a "build.bmp"} To resize a floating toolbar

- 1 Move the mouse pointer over any toolbar window border.
The mouse pointer turns into a sizing arrow.
- 2 Drag the border to resize the toolbar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Customizing the Keyboard

Keyboard shortcuts offer an alternate method of performing actions. With Microsoft Developer Studio, you can:

- Display the current keyboard shortcuts (this includes custom key settings and any selected editor emulation).
- Assign shortcut keys to available commands.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Displaying the Keyboard Shortcuts

You can display the current keyboard shortcuts, including custom key settings and editor emulations.

{ewl msdncd, EWGraphic, jug24g 0 /a "build.bmp"} To display keyboard shortcuts

- 1 From the Help menu, choose Keyboard.

The Help Keyboard dialog box appears, with a list of keyboard shortcuts.

- 2 Perform one or more of the following actions:

- Choose Category, Command, Keys, or Description to sort the list alphabetically in different ways.
- Choose Print to print out the contents of the dialog box.
- Choose Copy to copy the contents of the dialog box to the Clipboard so you can paste them into the word processor or editor of your choice.

- 3 Double-click the Control menu box in the upper-left corner to close the Help keyboard dialog box.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Assigning Shortcut Keys

You can use the Keyboard tab on the Customize dialog box to:

- Assign one or more shortcut keys to each of the available commands.
- Delete or change key assignments.
- Assign shortcut keys for each editor.
- Reset all shortcut keys to their default settings.

Note If you use any of the incremental search commands (IncrementalSearch, IncrementalSearchBack, IncrementalSearchRE, IncrementalSearchREBack), you can modify the search by toggling among the word (CTRL+W), regular expression (CTRL+T), and case sensitive (CTRL+C) modes. These keystrokes are not bindable and only affect the incremental search command.

{ewl msdncd, EWGraphic, jug25g 0 /a "build.bmp"} To assign a shortcut key

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Editor drop-down list box, select the editor that you want to assign the shortcut key to.
- 4 In the Categories list box, select the category containing the command that you want to assign the shortcut key to.
- 5 In the Commands list box, select the command to which you want to assign the shortcut key. A description of the command's effect appears in the Description box, and the currently assigned shortcut keys appear in the Current Keys box.
- 6 Click the Press New Shortcut Key box, and press the shortcut key or key combination that you want.

If you press a key or key combination that is invalid, no key is displayed, and the Assign button is unavailable. You cannot assign key combinations that use TAB, ESC, or F1, or any combination that uses CTRL+ALT+DEL, which Microsoft Windows NT uses.

If you press a key or key combination that is currently assigned to another command, that command appears under Currently Assigned To.
- 7 Choose Assign.

Any previous shortcut key assignment for the key or key combination that you specified is replaced by the new assignment.

You can repeat steps 3 through 7 until you have made all of the key assignments that you want.
- 8 Choose Close.

All of your shortcut key assignments are now in effect.

{ewl msdncd, EWGraphic, jug25g 1 /a "build.bmp"} To delete a shortcut key

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Editor drop-down list box, select the editor that has the shortcut key you want to delete.

- 4 In the Categories list box, select the category containing the command for which you want to delete the shortcut key.
- 5 In the Commands list, select the command for which you want to delete the shortcut key.
A description of the command's effect appears in the Description box, and the currently assigned shortcut keys appear in the Current Keys list box.
- 6 In the Current Keys list box, select the shortcut key to delete.
- 7 Choose Remove.
You can repeat steps 3 through 7 until you have deleted all of the key assignments that you want.
- 8 Choose Close.
All of your shortcut key deletions are now in effect.

{ewl msdncd, EWGraphic, jug25g 2 /a "build.bmp"} To reset all shortcut keys to their default values

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 Choose Reset All.
- 4 Choose Close.

All commands now have their original, default shortcut key assignments.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Customizing the Tools Menu

You can use the Tools tab in the Customize dialog box to add, delete, and edit Tools menu items. You can add frequently used utilities to the Tools menu and run them from within Microsoft Developer Studio.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding Commands to the Tools Menu

You can add up to 16 commands to the Tools menu. A tool can be any program that will run on your operating system.

As an example, the following procedure demonstrates how to add the Windows Notepad accessory to the Tools menu.

{ewl msdncd, EWGraphic, jug27g 0 /a "build.bmp"} To add a command to the Tools menu

- 1 From the Tools menu, choose Customize.

The Customize dialog box appears.

- 2 Select the Tools tab.

- 3 Choose Add.

The Add Tool dialog box appears.

- 4 In the Command text box, type `NOTEPAD.EXE`.

—or—

Choose Browse, select the appropriate drive and directory, and then select NOTEPAD.EXE from the list of filenames.

- 5 In the Arguments text box, type any arguments to be passed to the program.

You can use the drop-down arrow next to the Arguments text box to display a menu of arguments. Select an argument from the list to insert argument syntax into the Arguments text box.

- 6 In the Initial Directory text box, type the file directory where the command is located.

You can use the drop-down arrow next to the Initial Directory text box to display a menu of directories. Select a directory from the list to insert directory syntax into the Initial Directory text box.

- 7 Choose OK.

- 8 Choose Close.

The command now appears on the Tools menu. To run the program, choose it from the menu.

You can change the default menu name of the newly added tool by editing the Menu Text text box. You can also add arguments to be passed to the program by typing them in the Arguments text box (see Using Argument Macros), or set the initial directory for your program by typing it in the Initial Directory text box.

Note If the program you are adding to the Tools menu has a .PIF file, the startup directory specified by the .PIF file overrides the directory specified in the Initial Directory text box.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Editing a Tools Menu Command

{ewl msdncd, EWGraphic, jug28g 0 /a "build.bmp"} To edit a Tools menu command

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Tools tab.
- 3 In the Menu Contents box, select the menu command you want to edit.
- 4 Perform one or more of the following actions:
 - To move the selected command up one position in the menu, choose Move Up.
 - To move the selected command down one position in the menu, choose Move Down.
 - To change the menu text, command line (tool path and file name), command-line arguments, or the initial directory, type the new information in the appropriate text box.
 - To specify a letter in the menu title as an access key, precede that letter in the Menu Text text box with an ampersand (&).
 - The first letter in the title is the keyboard access key by default.
 - To be prompted for command-line arguments each time you run the tool, select the Prompt For Arguments check box.
- 5 Choose Close.

{ewl msdncd, EWGraphic, jug28g 1 /a "build.bmp"} To remove a command from the Tools menu

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Tools tab.
- 3 In the Menu Contents box, select the command you want to delete.
- 4 Choose Remove.
- 5 Choose Close button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Tools Options

The Tools tab in the Customize dialog box includes three check boxes for customizing options that apply to the tool currently selected in the Menu Contents box. These check boxes are described in the following table.

Option	Result
Program Arguments For Argumentum	When selected, the Tool Arguments <u>dialog box</u> appears when you run the tool. The arguments you type in the Arguments box are passed to the program.
Redirect Output To Output Window	When selected, the <u>standard output</u> from the tool appears in the Output window. Microsoft Developer Studio maintains a separate virtual Output window for each tool whose output has been redirected to the Output window. The names of these tools appear in a tab at the bottom of the Output window when you run them. To switch between virtual Output windows, select the tabs at the bottom of the Output window. See

[Using Error
Syntax for
Tools](#)

to learn about additional capabilities associated with redirecting tool output to the Output window.

Clo When selected,
se the command
Wi window
nd automatically
ow closes when
On the tool has
Exi finished
ting executing. This
option applies
to [character-
mode
applications](#)
only.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Argument Macros

You can specify arguments for any command that you add to the Tools menu.

{ewl msdn cd, EWGraphic, jug30g 0 /a "build.bmp"} To specify arguments for a Tools menu command

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Tools tab.
- 3 In the Menu Contents box, select the command for which you want to specify arguments.
- 4 In the Arguments text box, type the arguments that you want.
—or—
Select the drop-down arrow to the right of the Arguments text box to display a list of arguments.
Selecting an argument from this list, substitutes it as text in the Arguments text box.
- 5 Choose Close.

To help you integrate your tools with the environment, Developer Studio provides the argument macros shown in the following table.

Argument Macros

	Macro name	Expands to a string containing
	\$ (CursorColumn)	The cursor's current column position within the active window .
	\$ (CurrentDirectory)	The current working directory (defined as <i>drive+path</i>).
	\$ (CursorLine)	The cursor's current line position within the active window .
	\$ (CurrentText)	The current text (the word under the cursor's current position, or a single-line selection, if there is one).
	\$	The directory

(FileDir) of the current source (defined as *drive+path*); blank if a non-source window is active.

\$ The filename extension of the current source.

\$ The filename of the current source **(Filename)** (defined as *filename*); blank if a non-source window is active.

\$ The complete filename of the current source **(Filepath)** (defined as *drive+path+filename*); blank if a non-source window is active.

\$ The command-line arguments that are passed to the application you are developing. To set these command-line arguments, type the arguments in the Program Arguments text box on the Debug tab accessed by the Settings command on the Build menu.

\$ The directory of the current **(TargetDir)** [target](#) (defined as

drive+path).

\$ The filename
(Tar extension of
get the current
Ext) [target](#).

\$ The filename
(Tar of the current
get [target](#)
Na (defined as
me) *filename*).

\$ The complete
(Tar filename of the
get current [target](#)
Pat (defined as
h) *drive+path+file
name*).

\$ The directory
(Wk of the current
spD [workspace](#)
ir) (defined as
drive+path)
that contains
the .MDP file;
blank if no
[workspace](#)
is currently
open.

\$ The current
(Wk [workspace](#)
spN name (defined
am as *filename*)
e) without
the .MDP
extension;
blank if no
[workspace](#)
is currently
open.

Macro recognition is not case sensitive. All path macros end in a backslash (\).

To use a macro as an argument, type the macro name in the Arguments text box. Or, for macros that expand to a directory, type the macro name in the Initial Directory box. As an example, the following procedure demonstrates how to add the **\$(FilePath)** argument macro to the Windows Notepad accessory (installed in a previous procedure).

{ewl msdncd, EWGraphic, jug30g 1 /a "build.bmp"} To add an argument macro to an installed tool and then run it

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Tools tab.
- 3 In the Menu Contents box, select the command that you want to edit.

In this case, select the Notepad accessory that you installed earlier.

- 4 In the arguments text box, type `$(FilePath)`.

–or–

Select the drop-down arrow to the right of the Arguments text box to display a list of Arguments.

When you select an argument from this list, the argument is substituted as text in the Arguments text box.

For example, use the drop-down arrow to the right of the Arguments text box and select the **\$ (FilePath) macro**.

- 5 Choose Close.
- 6 Open any source file, or make an open source file active by clicking it.
- 7 From the Tools menu, choose Notepad.

The Windows Notepad accessory opens, with the active source file as its text file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Error Syntax for Tools

When you select the Redirect To Output Window check box, you gain access to the Output window's error parser.

The error parser detects filenames, errors, and line-number information of output strings, and makes each line in the file a hot link to the specified file and line number. For example, you can double-click an error line in the Output window that contains the error number, filename, and line number where the error occurred, and jump directly to the referenced line in the correct source file.

The Find In Files dialog box also uses the error parser. For instance, you can double-click any output line from a Find In Files operation to jump to the referenced file and line.

Error Syntax Example

For example, you could install Microsoft Macro Assembler on the Tools menu to compile assembly code, and then jump to source-code syntax errors directly from its error list in the Output window. The error syntax is as follows (+ denotes one or more; * denotes zero or more):

Error Type	Description
error_string	file_spec error_spec (STRING file_spec STRING)
file_spec	FILENAME ('line_spec') ':'
line_spec	NUMBER NUMBER ' <u>'</u> NUMBER
error_spec	ERRORKEYWORD ERRORNUMBER ':'
where	
:	
STRING	Null-terminated string
FILENAME	Valid file specification and text file
NUMBER	{1-9}{0-9}
ERROR	{A-Z}{0-9}{0-9}

ORN 9}{0–9}
UMB
ER
"error" |
ERR "warning" |
ORK "fatal error"
EYW
ORD

Note Although the error number is part of this syntax, it is optional and not really useful to any tool except internal build tools. The error number is used internally to link to Books Online.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Showing the Status Bar

The status bar at the bottom of the application window displays information about Developer Studio. Its leftmost text field, for instance, describes the currently selected menu command or the action of the button currently under the mouse pointer.

The status bar also displays progress information about the current operation. In a text editor window, it shows the line and column position of the insertion point, the state of the RECORD KEYSTROKES AND COLUMN MODE, whether the editor is in insertion mode or overstrike mode, and whether the file is set for read-only access. Optionally, the clock can also be displayed on the status bar. Figure 14.8 depicts the status bar as it might appear while using the text editor.

Figure 14.8 A Status Bar

{ewc msdncd, EWGraphic, jug33g 0 /a "uguideSTAT.BMP"}

The default setting is to show the status bar.

{ewl msdncd, EWGraphic, jug33g 1 /a "build.bmp"} To show or hide the status bar

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Workspace tab.
- 3 Select the Display Status Bar check box to show the status bar, or clear the Display Status Bar check box to hide the status bar.
- 4 Choose OK.

{ewl msdncd, EWGraphic, jug33g 2 /a "build.bmp"} To display the clock on the status bar

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Workspace tab.
- 3 Select the Display Clock On Status Bar check box.
- 4 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Directories

The Visual J++ and Java Virtual Machine Setup programs determine default directory and class paths for several file types. These file types are:

- Executable files (build utilities)
- Class files
- Source files

Note The path associated with the class files is also known as the classpath. Entries made for class files on the Directories tab are appended to the class path information registered by the Java Virtual Machine and the CLASSPATH environment variable. See CLASSPATH environment variable for more information.

On the Directories tab, accessed by clicking Options from the Tools menu, you can edit the directory paths where Microsoft Developer Studio and the Java Virtual Machine looks for specific file types.

Directory information is stored in registry entries. The Show Directories For list box on the Directories tab displays the directories shown in the following table.

File Path type contents	
Exe cuta ble files	Specifies where JVC, JVIEW and other utilities reside.
Clas s files	Specifies additional directories to search for class files (files referenced with the Java import statement) that are not included in the default setting. Class paths entered will be searched after the directories specified by the class path information registered by the Java Virtual Machine and the

CLASSPATH Environment variable.

The default setting for class files is determined by the JAVA VM class path value in the registry and the CLASSPATH environment variable.

Developer Studio does not display the default setting on the Directories tab.

Source Specifies where source code files reside.

{ewl msdncd, EWGraphic, jug34g 0 /a "build.bmp"} To add a directory to the Directories list

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Directories tab.
- 3 If necessary, select the platform from the Platform list box.
- 4 In the Show Directories For list box, select the category of directory.
- 5 In the Directories box, double-click the blank line at the bottom of the list (indicated by an empty rectangle), and type the directory name.
- 6 Choose OK.

Developer Studio searches directories in the order in which they appear in the list. After adding a directory, you can move it up or down in the list by dragging it and dropping it in the new position.

{ewl msdncd, EWGraphic, jug34g 1 /a "build.bmp"} To remove a directory from the Directories list

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Directories tab.
- 3 If necessary, select the platform from the Platform list box.

- 4 In the Show Directories For list box, select the category of directory.
- 5 In the Directories list box, double-click the directory that you want to remove.
- 6 Delete the text defining the directory.
- 7 Choose OK.

{ewl msdncd, EWGraphic, jug34g 2 /a "build.bmp"} To prioritize a directory in the Directories list

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Directories tab.
- 3 If necessary, select the platform from the Platform list box.
- 4 In the Show Directories For list box, select the category of directory.
- 5 In the Directories box, select the directory that you want to prioritize.
- 6 Drag the selected directory to its new position.
- 7 Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Full-Screen Mode

You can use full-screen mode with the text editor and other resource editors. To toggle full-screen mode on and off, click on the full-screen-mode toolbar button—a small graphic of a computer screen that initially appears in the upper-left corner of the screen. Like any other toolbar, this toolbar button can be moved and, each time you select full-screen mode, will appear in the location you last left it in. If you close this toolbar button and want to restore it, follow the procedure below to redisplay this button.

{ewl msdncd, EWGraphic, jug35g 0 /a "build.bmp"} To begin full-screen mode

- From the View menu, choose Full Screen.

When you switch to full-screen mode for the first time, your current standard-mode horizontal and vertical scroll bar settings are used for full-screen mode. However, you can have different window settings for full-screen mode and standard mode.

{ewl msdncd, EWGraphic, jug35g 1 /a "build.bmp"} To change the full-screen mode window settings

- 1 Begin full-screen mode.
- 2 From the Tools menu, choose Options.
When full-screen mode is active, press ALT+T to display the Tools menu.
- 3 Select the Editor tab.
- 4 Check the Window Settings that you want.
- 5 Choose OK.

{ewl msdncd, EWGraphic, jug35g 2 /a "build.bmp"} To end full-screen mode

- Press ESC.
—or—
Click the full-screen toolbar button.
—or—
Press ALT+V to display the View menu and choose Full Screen.

{ewl msdncd, EWGraphic, jug35g 3 /a "build.bmp"} To redisplay the full-screen toolbar button while in full-screen mode

- 1 Begin full-screen mode.
- 2 From the Tools menu, choose Customize.
When full-screen mode is active, press ALT+T to display the Tools menu.
- 3 Select the Toolbars tab.
- 4 In the Categories list box, select View.
- 5 Drag the Toggle Full Screen button onto the full-screen application window.
- 6 Choose Close.

Note Open a file for editing before beginning full-screen mode. If you do not open a file first, full-screen mode appears as a large, empty screen. If this happens, use the ESC key to restore the original screen mode.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Customizing with Other Options

You can choose other options to customize editing, debugging, working with projects.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Editor Emulations

The Microsoft Developer Studio text editor can emulate two popular text editors: BRIEF® and Epsilon™. With the emulation feature, the text editor can emulate the key bindings, text selection, caret display, window display, and most editing commands of the selected editor. Some editor behaviors are not available, notably those dealing with macros, shells, and other elements that have no substitute in the text editor.

The Epsilon emulation is based on the Lugaru Epsilon editor version 6.0. The BRIEF emulation is based on the Borland BRIEF editor version 3.1.

Note Each editor emulation includes the use of native syntax for regular expressions during find and replace operations. For more information on regular expression syntax, see [Using Regular Expressions with Developer Studio](#), [Using Regular Expressions with BRIEF Emulation](#), and [Using Regular Expressions with Epsilon Emulation](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setting Editor Behavior

You can use the Compatibility tab in the Options dialog box to set overall editor behavior. The Compatibility tab contains a drop-down list box of the available editors for emulation. The supported editor emulations are:

- Developer Studio
- Visual C++ version 2.0
- BRIEF
- Epsilon

The Options checklist contains the compatibility options and their default settings for the chosen editor. You can change these options to create a custom emulation model. When you create a custom emulation model, the word “Custom” appears in the list box with the name of the standard editor. For example, if you change some of the options for the BRIEF emulation, “Custom (BRIEF)” appears in the list.

For each emulation, the following default options are set:

Developer Studio

- Enable copy without selection

Visual C++ version 2.0

- Enable copy without selection
- Enable virtual space

BRIEF

- Disable backspace at start of line
- Enable copy without selection
- Enable line-mode pastes
- Enable virtual space
- Include caret positioning in undo buffer
- Use BRIEF's regular expression syntax

Epsilon

- Include caret positioning in undo buffer

{ewl msdncd, EWGraphic, jug1h 0 /a "build.bmp"} To set an editor emulation

- 1 From the Tools menu, choose Options.
The Options dialog box appears.
- 2 Select the Compatibility tab.
- 3 In the Recommended Options For list box, select the editor that you wish to emulate.
The default editor is Developer Studio.
The Options box lists the status of predefined editor options.
- 4 Choose the OK button.

{ewl msdncd, EWGraphic, jug1h 1 /a "build.bmp"} To create a custom editor

emulation

- 1** From the Tools menu, choose Options.
The Options dialog box appears.
- 2** Select the Compatibility tab.
- 3** In the Recommended Options For list box, select a standard editor on which to base your custom editor.
The Options box lists the editor's current options.
- 4** Select the options you want to create the desired editor behavior.
The name of the custom editor reflects the name of the standard editor. For example, if you customize the BRIEF emulation, the custom editor is named "Custom (BRIEF)".
- 5** Choose the OK button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using Epsilon Emulation

The Epsilon emulation provides Epsilon default key bindings, caret display, text selection, and the following general editing commands.

Tip You can change individual shortcut keys with the Keyboard tab in the Customize dialog box.

Epsilon Developer Studio		
Cat ego ry	Comm and	per Studio Comm and
Help	help	(Command on help function)
Bookmarks	set-bookmarks	BookmarkDrop (Epsilon)
	jump-to-last-bookmark	BookmarkJumpToLast
	set-named-bookmark	Bookmark
	jump-to-named-bookmark	Bookmark
Buffer	select-buffer	Window List
Files	find-files	FileOpen
	save-file	FileSave
	write-file	FileSaveAs
	insert-file	InsertFile
	save-all-buffers	FileSaveAs
Indentation	to-indentation	GoToIndentation

	indent-previous	(Use the TAB key.)
	indent-region	IndentSelectionToPrev
	center-line	WindowScrollToCenter
	tabify-region	TabifySelection
	untabify-region	UntabifySelection
	indent-under	IndentToPrev
Inserting and Deleting	quoted-insert	QuotedInsert
Opening, Backward Deleting, Deleting Space, Overwriting, and Macros	open-line	LineOpenAbove
	backward-delete-character	(Use the BACKSPACE key.)
	delete-character	Delete
	delete-horizontal-space	DeleteHorizontalSpace
	delete-blank-lines	DeleteBlankLines
	overwrite-mode	(Use the INSERT key)
	Keymap	ToolsRecordKeys
	end-	ToolsSt

	kbd-macro	opRecording
	last-kbd-macro	ToolsPlaybackRecording
Killing and Yanking	set-mark	StreamSelectExclusive
	highlight-region	SelectHighlight
	exchange-point-and-mark	SelectSwapAnchor
	kill-line	LineCut
	kill-region	CutSelection
	copy-region	Copy
	yank	Paste
	append-next-kill	AppendNextCut
	rectangle-mode	SelectColumn
Miscellaneous	abort	Cancel
	exit	FileExit
	argument	SetRepeatCount
	goto-line	GoTo
Moving Around	beginning-of-line	Home
	end-of-line	LineEnd
	down-line	LineDown
	up-line	LineUp
	forward	CharRight

-	ght	
charact		
er		
backwa	CharLef	
rd-	t	
charact		
er		
center-	Window	
window	ScrollTo	
	Center	
next-	PageDo	
page	wn	
previou	PageUp	
s-page		
scroll-	Window	
up	ScrollU	
	p	
scroll-	Window	
down	ScrollD	
	own	
goto-	Docum	
beginni	entStart	
ng		
goto-	Docum	
end	entEnd	
beginni	Window	
ng-of-	Start	
window		
end-of-	Window	
window	End	
Par	forward	ParaDo
agr -	wn	
aph	paragra	
s	ph	
	backwa	ParaUp
	rd-	
	paragra	
	ph	
	mark-	SelectP
	paragra	ara
	ph	
Par		
ent	find-	GoToM
heti	delimite	atchBra
c	r	ce
Exp		
res		
sio		
ns		
	forward	LevelD
	-level	own

	backwa	LevelU
	rd-level	p
	kill-level	LevelC
		utToEn
		d
	backwa	LevelC
	rd-kill-	utToSta
	level	rt
Ru	next-	GoToN
nni	error	extError
ng		Tag (A
Pro		default
gra		key
ms		binding
		is not
		provide
		d for
		this
		comma
		nd.)
	previou	GoToPr
	s-error	evError
		Tag (A
		default
		key
		binding
		is not
		provide
		d for
		this
		comma
		nd.)
Sen	forward	Senten
ten	-	ceRight
ces	sentenc	
	e	
	backwa	Senten
	rd-	ceLeft
	sentenc	
	e	
	kill-	Senten
	sentenc	ceCut
	e	
Sea		
rchi	increme	Increme
ng	ntal-	ntalSea
and	search	rch
Re		
pla		
cin		
g		
	reverse	Increme
	-	ntalSea
	increme	

	ntal- search	rchBack
	regex- search	Increme ntalSea rchRE
	reverse -regex- search	Increme ntalSea rchREB ack
	grep	FileFind InFiles
	next- match	FindNe xt
	previou s-match	FindPre v
	replace- string	FindRe place
	query- replace	FindRe place
	regex- replace	FindRe placeR E
	word- mode, regular- express ion- mode, case- sensitiv e- mode, and increme ntal- mode	(These comma nds are availabl e only in increme ntal search mode, not in dialog mode. The comma nds are not key bindabl e.)
Tag	goto- tag	Browse
s	pluck- tag	Browse GoToD efinition
Tra	transpo	CharTra
nsp	se-	nspose
osi	charact	
ng	ers	
	transpo	WordTr
	se-	anspos

	words	e
	transpo	LineTra
	se-lines	nspose
Un	undo	Undo
do		
	redo	Redo
	undo-	UndoC
	change	hanges
	s	
	redo-	RedoC
	change	hanges
	s	
Win	one-	Window
do	window	SingleP
ws		ane
	split-	Window
	window	SplitHor
		izontal
	split-	Window
	window	SplitVer
	-	tical
	verticall	
	y	
	kill-	Window
	window	KillPan
		e
	zoom-	Window
	window	Maximi
		ze
	move-	Window
	to-	NextPa
	window	ne
	next-	Window
	window	Cycle
	previou	Window
	s-	Previou
	window	s
Wor	forward	WordRi
d	-word	ght
Co		
mm		
and		
s		
	backwa	WordLe
	rd-word	ft
	backwa	WordD
	rd-kill-	eleTo
	word	Start
	kill-	WordD
	word	eleTo
		End

transpo	WordTr
se-	anspos
words	e
capitaliz	WordC
e-word	apitaliz
	e
lowerca	WordLo
se-word	werCas
	e
upperca	WordU
se-word	pperCa
	se

Note The entire set of emulation commands is available to each editor. For more information, see [Viewing and Changing the Shortcut Keys](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using BRIEF Emulation

The BRIEF emulation provides BRIEF default key bindings, caret display, text selection, and the following general editing commands.

Tip You can change individual shortcut keys with the Keyboard tab in the Customize dialog box.

	BRIEF Category	Command	Developer Studio Command
	Help	Help	(Command on help function)
	Undo and Redo	Undo	Undo
		Redo	Redo
	Saving and Exiting	Exit	FileExit
		Write	FileSave
		Write All and Exit	FileSaveAllExit
	Editing Text	Backspace	(Use the BACKSPACE key)
		Delete	(Use the DELETE key)
		Delete Line	LineDelete
		Delete Next Word	WordDeleteToEnd
		Delete Previous Word	WordDeleteToStart

	Delete to Beginning of Line	LineDeleteToStart
	Delete to End of Line	LineDeleteToEnd
	Enter	(Use the ENTER key)
	Insert Mode Toggle	EditToggleOverType
	Open Line	LineOpenBelow
	Quote	QuotedInsert
Buffers	Delete Current Buffer	FileClose
	Edit File	FileOpen
	Next Buffer	WindowNext
	Previous Buffer	WindowPrevious
	Read File into Buffer	InsertFile
Search and Translate	Case Sensitivity Toggle	ToggleCaseSensitivity
	Incremental Search	IncrementalSearch
	Regular Expressions Toggle	EditToggleRE
	Search Again	FindRepeat
	Search Backward	FindPrevious

	Search	Find
	Forward	
	Translate	FindRepeat
		Again place
	Translate	FindRepeat
		place
	Forward	
Windows	Center Line in Window	Window ScrollTo Center
	Change Window	Window SwitchPaneUp, Window SwitchPaneDown, Window SwitchPaneLeft, Window SwitchPaneRight. (A default key binding is not provided for this command.)
	Create Window	Window SplitHorizontal, Window SplitVertical
	Delete Window	Window DeleteRowUp, Window DeleteRowDown, Window DeleteColumnLeft, Window DeleteColumnRight

Line to Bottom of Window	Window ScrollTo Bottom
Line to Top of Window	Window ScrollTo Top
Quick Window Switch	Window SwitchP aneUp, Window SwitchP aneDown, Window SwitchP aneLeft, Window SwitchP aneRight
Resize Window	Window Split
Zoom Window Toggle	Window Maximize
Blocks and Marks	Column SelectColumn Mark
Drop Bookmark	BookmarkDrop (BRIEF)
Indent Block	IndentSelection
Jump Bookmark	Bookmark
Line Mark	SelectLine
Lower Case Block	LowerCaseSelection
Mark/Unmark	SelectCharacter
Noninclusive Mark	SelectCharacterInclusive
Outdent Block	UnindentSelection

	Print Block	FilePrint
	Swap Cursor and Mark	SelectSwapAnchor
	Upper Case Block	UpperCaseSelection
Scrap	Copy to Scrap	Copy
	Cut to Scrap	Cut
	Paste from Scrap	Paste
Cursor Movement	Back Tab	(Use the SHIFT+TAB keys)
	Beginning of Line	Home(BRIEF)
	Cursor Movement	(Use the arrow keys)
	End of Buffer	DocumentEnd
	End of Line	End(BRIEF)
	End of Window	WindowEnd
	Go to Line	GoTo
	Left Side of Window	WindowLeftEdge
	Next Character	(Use the right arrow key)
	Next Word	WordRight
	Page Down	PageDown
	Page Up	PageUp
	Previous	(Use

s	the left
Character	arrow
er	key)
Previous	WordLe
s Word	ft
Right	Window
Side of	RightEd
Window	ge
Scroll	Window
Buffer	ScrollU
Down in	p
Window	
Scroll	Window
Buffer	ScrollD
Up in	own
Window	
Tab	(Use
	the TAB
	key)
Top of	Home(
Buffer	BRIEF)
Top of	Window
Window	Start

Ma
cro
s, Pause ToolsPa
Pla Recordi useRec
yba ng ording
ck, Toggle
and
Re
me
mb
er

Playbac	ToolsPI
k	ayback
	Recordi
	ng
Remem	ToolsRe
ber	cordKe
	ystroke
	s

Spe
cial Go To Browse
Co Routine
mm
and
s

Next	GoToN
Error	extError
	Tag
Repeat	EditSet

Repeat
Count

Note The entire set of emulation commands is available to each editor. For more information, see [Viewing and Changing the Shortcut Keys](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Viewing and Changing the Shortcut Keys

You can customize the keyboard for the selected editor and the editing commands. Customization is stored only for the current editor. If you change emulation, you will lose all of your custom key assignments.

Specific commands and their associated shortcut keys are available for the following categories:

- File
- Edit
- View
- Insert
- Build
- Debug
- Tools
- Image
- Layout
- Window
- Help

Each category contains a variety of commands that you can assign to individual keystrokes. For more information, see Customizing the Keyboard.

Note You can display all of the current keyboard shortcuts, including custom key settings and editor emulations. For more information, see Displaying the Keyboard Shortcuts.

{ewl msdn cd, EWGraphic, jug8h 0 /a "build.bmp"} To find the current shortcut key

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Editor drop-down list box, select Text.
- 4 In the Categories list box, select the category.
- 5 In the Commands list box, select the command.
Epsilon-specific commands contain the text "Epsilon." BRIEF-specific commands contain the text "BRIEF."
The Current Keys box displays the current shortcut keys. Multiple assignments are listed on separate lines.
- 6 Choose the Close button.

{ewl msdn cd, EWGraphic, jug8h 1 /a "build.bmp"} To change the current shortcut key

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Categories list box, select the category.
- 4 In the Commands list box, select the command.

The Current Keys box displays the current shortcut keys. Multiple assignments are listed on separate lines.

- 5 Change the focus to the Press New Shortcut Key box, then press the keystroke combination you want to assign. If you make a mistake, use the BACKSPACE key to correct it, not the DEL key. If the key combination is currently assigned to another command, the command is displayed in the Currently Assigned To box.
- 6 Choose the Assign button.
The new keystroke combination is added to the Current Keys box.
- 7 Choose the Close button.

{ewl msdncd, EWGraphic, jug8h 2 /a "build.bmp"} To remove a shortcut key assignment

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Categories list box, select the category.
- 4 In the Commands list box, select the command.
The Current Keys box displays the current shortcut keys. Multiple assignments are listed on separate lines.
- 5 Select the current key assignment that you want to remove.
- 6 Choose the Remove button.
The keystroke combination is removed from the Current Keys box.
- 7 Choose the Close button.

{ewl msdncd, EWGraphic, jug8h 3 /a "build.bmp"} To reset all keystroke assignments

- 1 From the Tools menu, choose Customize.
The Customize dialog box appears.
- 2 Select the Keyboard tab.
- 3 In the Editor drop-down list box, select the editor.
- 4 Choose the Reset All button and confirm your choice.
All keystroke combinations for the selected editor revert to their default settings.
- 5 Choose the Close button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

JVC Reference

JVC.EXE (JVC) is the Microsoft Java compiler. The compiler produces interpreted (.CLASS) files that will run on any Java Virtual Machine.

Some more common compiler options are easily accessed from the Java tab on the Project Settings dialog box; all options may be typed into the Project Options text box of the same tab. Each of the options on the Java tab is described in Setting Compiler Options. The description of each option includes the name of the equivalent command-line option.

For an alphabetic reference to the JVC options, see Reference to JVC Command-Line Options. Other topics covered include:

- Description of JVC Syntax
- Using JVC.EXE

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Description of JVC Syntax

The JVC.EXE (JVC) command line uses the following syntax:

JVC [*options*] <*filename*>

The following table describes input to the JVC command.

Ent	Meaning
ry	
<i>option</i>	One or more JVC options. See Setting Compiler Options , and Reference to Command-Line Options for more information. Note that all options apply to all specified source files.
<i>filename</i>	The name of one or more source files.

You can specify any number of options and filenames, as long as the number of characters on the command line does not exceed 1024 or the limit dictated by the operating system.

Note There is no guarantee that future releases of Microsoft Windows NT (version 4.0 and later) and Microsoft Windows 95 will have the same input limit of 1024 characters for the command line.

Any options that you wish to pass to JVC must be supplied before the name of the .CLASS file or they will be interpreted as command-line arguments to the .CLASS file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Filename Syntax

The JVC recognizes the following filename syntax:

- JVC accepts files with names that follow FAT, HPFS, or NTFS naming conventions.
- Any filename can include a full or partial path.

A **full path** includes a drive name and one or more directory names. JVC accepts filenames separated either by backslashes (\) or forward slashes (/).

A partial path omits the drive name, which JVC assumes to be the current drive. If you don't specify a path, JVC assumes the file is in the current directory.

Note If a file does not have an extension, JVC assumes the extension of .JAVA.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using JVC.EXE

You can use JVC.EXE (JVC) to compile specified .JAVA source files into interpreted .CLASS files. To compile without producing the .CLASS files, use the /nowrite option.

You can specify JVC options on the command line, or in Developer Studio. If you are compiling your Java project from within Developer Studio, specify the JVC options you wish to use in the Project Options text box of the Java tab from the Build menu's Project Setting dialog box.

If you are compiling your Java project from the command line, place the options you wish to supply to JVC before the name of your .CLASS file on the command line.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Order of JVC Options

Command-line options can appear anywhere on the JVC.EXE (JVC) command line. JVC reads the command line from left to right processing command files in the order it encounters them. Each option applies to all files on the command line. If JVC encounters conflicting options, it uses the rightmost option.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


JVC Command Files

A command file, also referred to as a response file, is a text file that contains information you would otherwise type on the command line. JVC accepts a command file as an argument on the command line. Unlike the command line, a command file allows you to use multiple filenames.

The criteria listed below should be followed when using command files:

- A command file may contain only filenames.
- A command file must not invoke the JVC command.
- A command file must not contain any JVC compiler options.

A command file is specified by an “at” sign (@) followed by a filename; the filename can specify an absolute or relative path.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CLASSPATH Environment Variable

Use the CLASSPATH environment variable to specify additional class path information when compiling from the command line. The CLASSPATH environment variable has the following syntax:

SET CLASSPATH = <path>

An understanding of where the Virtual Machine (VM), Visual J++, and the command line get classpath information will explain why your Java program may behave differently depending upon where it executes.

When the Virtual Machine (VM) installs, it enters the value C:\Windows\Java\Classes\Classes.zip;.; into the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM\Classpath. When a Java program runs, Visual J++ and the VM first look for the CLASSPATH value in the registry. If the application executes from the command line, Visual J++ and the VM also use the CLASSPATH environment variable if one has been set. When execution launches from Microsoft Developer Studio, they also search the directories listed in the Directories tab on the Options dialog of the Tools menu. See Setting Directories for more information.

What this means is if you are executing from the:

- Command line:
Classpath = C:\Windows\Java\Classes\Classes.zip;.; and the value set in the CLASSPATH environment variable
- Developer Studio:
Classpath = C:\Windows\Java\Classes\Classes.zip;.; and the value set in the Directories tab of the Options dialog from the Tools menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Reference to JVC Command-Line Options

This topic is an alphabetic reference to all the JVC command-line options. These options are listed in Table 16.1.

If a command-line option can take one or more arguments, its syntax is shown under a Syntax heading before its description. Click any option in the following table for information on the option.

Table 16.1 JVC Options Set from the Command Line

/cp	/g:l	/O:J
/cp:o	/g:t	/
		_
		verbo
		se
/cp:p	/	/w
	nowri	
	te	
/d	/O	/?
/g	/O:l	

All options listed in Table 16.1 may be typed into the Common Options text box on the Java tab of the Project Settings dialog box. These options can also be used from the command line.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/cp

Syntax

/cp classpath

Use the /cp option to set the CLASSPATH environment variable for the current compilation. Using this option specifies the path where the JVC or JView command-line tools can find system and user-defined classes. The Java interpreter uses a platform-dependent default location and the CLASSPATH environment variable to find system classes. (See CLASSPATH Environment Variable for more information.) The directories in the class path are separated by semicolons on a Microsoft Windows system; by colons on Unix.

Example

For example, on Windows NT, the class path might be:

```
JVC /cp x:.;x:\java\classes
```

In this example JVC searches in and beneath the directories on the path for system and user-defined classes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


`/cp:o`

This option prints the class path to standard output. This option is especially useful for troubleshooting errors, like “class not found,” when compiling from the command line.

Example

The following example prints the class path to the screen.

```
JVC /cp:o
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/cp:p

Syntax

/cp:p *path*

This option prepends the *path* entered to the *class path* and inserts a semicolon between them. When multiple /cp:p switches are entered, the paths are concatenated.

Note The value for the class path may come from the CLASSPATH environment variable or from the /cp option.

```
JVC /cp:p myproj
```

When multiple /cp:p switches are entered, the paths are concatenated.

Example

The following command concatenates the directories, myproj1 and myproj2, and prepends the resulting path to the existing class path:

```
JVC /cp:p myproj1 /cp:p myproj2
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/d

Syntax

/d directory filename

When compiling .JAVA files, use the /d option to specify an output directory other than the current directory for the .CLASS files. If the directory does not exist, JVC will create it.

Example

The following command compiles the myClass.java file into a myClass.class file and writes this file into the class directory.

```
JVC /d c:\classdir myClass.java
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/g

Syntax

/g[-] *filename*

The /g option generates all debugging information. No debugging options are set by default. The effect of using the /g switch is the same as using the following options together:

Option	Action
<u>/g:l</u>	generate line number information
<u>/g:t</u>	generate debug tables

Note To disable this option, use a dash (-) after the switch on the command line.

Examples

The following command creates a .CLASS file called myClass.class that contains debugging information:

```
JVC /g myClass.java
```

The following command instructs JVC to exclude debugging information from the myClass.class file:

```
JVC /g- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

`/g:l`

Syntax

`/g:l[-] filename`

This option generates line numbers that are used when debugging an program. No debugging options are set by default.

Note To disable this option, use a dash (–) after the switch on the command line.

Examples

The following command instructs JVC to generate line-number information for the resulting `myClass.class` file:

```
JVC /g:l myClass.java
```

The following command instructs JVC to turn off line-number generation:

```
JVC /g:l- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/g:t

Syntax

/g:t[-] *filename*

This option generates debug tables that are used when debugging an program. No debugging options are set by default.

Note To disable this option, use a dash (–) after the switch on the command line.

Examples

The following command instructs JVC to generate debug-table information for the resulting myClass.class file:

```
JVC /g:t myClass.java
```

The following command instructs JVC to turn off debug-table generation:

```
JVC /g:t- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/nowrite

Syntax

/nowrite filename

The /nowrite switch tells JVC to compile a .JAVA file and to suppress the writing of a .CLASS file.

Example

In the following example, the myClass.java file is compiled, errors and warnings are reported, but no myClass.class file is produced:

```
JVC /nowrite myClass.java
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/O

Syntax

/O filename

The /O option combines optimizing options to produce the fastest possible program. The effect of using this option is the same as using the following options together:

Option	Action
<u>/O:i</u>	Optimize by inlining methods when appropriate
<u>/O:j</u>	Optimize bytecode jumps. Default optimization setting.

Note To disable any optimization option, use a dash (–) after the switch on the command line.

Examples

The following command fully optimizes the code for myClass.class file that is produced:

```
JVC /O myClass.java
```

The following command turns off all code optimization:

```
JVC /O- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/O:I

Syntax

/O:I filename

The /O:I option tells the compiler it may inline methods to produce more efficient code. Code that is inlined does not have the overhead associated with a method call. Since there is no mechanism in the Java language to request inlining of methods, use this option when you want the compiler to inline your code.

Note To disable this option, use a dash (–) after the switch on the command line.

Examples

The following example evaluates the source code in the myClass.java file and inlines methods where possible. The resulting myClass.class file contains the optimized code.

```
JVC /O:I myClass.java
```

The following command turns off all inline-code optimization:

```
JVC /O:I- myClass.java
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/O:J

Syntax

/O:J filename

The /O:J option causes JVC to optimize the bytecode jumps in the compiled .CLASS file. This switch tells the compiler to generate code that jumps to another jump. This switch is the compiler's default optimization setting.

Note To disable this option, use a dash (–) after the switch on the command line.

Examples

The following example evaluates the source code in the myClass.java file and optimizes the bytecode jumps to produce more efficient code. The resulting myClass.class file contains the optimized code.

```
JVC /O:J myClass.java
```

The following command turns off bytecode jump optimization:

```
JVC /O:J- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/verbose

Syntax

/verbose *filename*

This option instructs JVC to display all messages while compiling a file or project. These messages give useful information about the progress of the compilation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/w

Syntax

/w0

/w1

/w2

/w3

/w4

These options control the number of warning messages produced by the compiler. They affect only source files named on the command line.

Compiler warning message numbers begin with J5. The documentation describes the warnings, indicates each warning's level, and indicates potential problems (rather than actual coding errors) with statements that may not compile as you intend.

The meaning of the options is as follows:

Option	Description
/w0	Turns off all warning messages.
/w1	Displays severe warning messages.
/w2	Displays a less-severe level of warning <u>message</u> than /w1. This is the default.
/w3	Displays a less-severe level of warning <u>message</u> than /w2, including use of methods with no declared return type, failure to put return statements in methods with nonvoid return types, and data conversions that would cause loss

of data or
precision.

/ Displays the
w least severe
4 level of warning
messages.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/x

Syntax

`/x[-] filename`

The Visual J++ compiler supports the Java language specification. In addition, it may offer a number of features beyond those required by this specification. These features are available by default, and are not available when the /x option is specified.

Use /x if you plan to port your program to other environments. The /x option tells the compiler to treat extended keywords as simple identifiers and to disable the other Microsoft extensions.

Note To disable this option, use a dash (–) after the switch on the command line.

Examples

The following command ignores any Visual J++ extensions used in the myClass.class file:

```
JVC /x myClass.java
```

The following command restores the default settings and recognizes the Visual J++ language extensions:

```
JVC /x- myClass.java
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

/?

Syntax

/?

This option displays a listing of compiler options to standard output.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Initializing and Configuring Microsoft Developer Studio

Microsoft Developer Studio stores information about initialization and configuration within the Registry. Most of the settings in the Registry are read-write: Developer Studio reads them at startup and writes them at the end of the session if they have changed. Other settings are read-only: Developer Studio reads them at startup but never writes them. The only way that you can change this read-only information is to use the Registry editor. For Microsoft Windows NT, the Registry editor is REGEDT32.EXE, and for Microsoft Windows 95, the Registry editor is REGEDIT.EXE.

The Registry is divided into “keys,” which are represented as folders in the Registry editor. Each key contains one or more entries, consisting of a value and a string or number, which are shown in the right pane of the Registry editor. For example, the Dialog Editor key in the Registry contains information about the startup settings for Grid and GridSize.

Caution Microsoft Developer Studio and other applications use Registry information to control each application’s behavior. If the expected Registry information is missing or incorrect, the application’s behavior may be unpredictable. When you modify or add Registry keys, be sure to enter the key and its values correctly.

You can customize Developer Studio by modifying existing Registry information and adding various device descriptions and default settings. For any of these changes to become effective however, you must first close down, and then restart Developer Studio.

{ewl msdnecd, EWGraphic, jug0i 0 /a "build.bmp"} To modify Registry information

- 1 From the MS-DOS prompt, run REGEDIT.EXE (for Windows 95) or REGEDT32.EXE (for Windows NT).

–or–

Double-click the Registry icon in the system tools program group.

The Registry editor appears.

Note You may need to add the Registry icon to your system tools program group.

- 2 Select the folder with the Registry information you want to modify.
- 3 To open a Registry key for editing, double-click the Registry key.
–or–
From the Edit menu, choose Modify, Delete, or New.
If you select New, you also select the type of new key: Key, String Value, Binary Value, or DWORD Value.
- 4 Type the Value Name, Value Data, and Base (either hexadecimal or decimal).
- 5 Choose OK.

For more information on defining Registry keys, see the following examples:

- [Setting Default Dialog Box Buttons](#)
- [Setting User Interface Fonts](#)
- [Setting the Default Magnification Factor](#)
- [Describing Mouse Pointer Devices](#)
- [Describing Icon Devices](#)


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting Default Dialog Box Buttons

Key Name: HKEY_CURRENT_USER\Software\Microsoft\Developer\Dialog Editor

Value Name: InitialButtons

Data Type: REG_DWORD

Data: *integer*

This key determines if a new dialog box template is created with the OK and Cancel buttons, where:

- If *integer* is 0, the dialog box templates are blank.
- If *integer* is 1, the dialog box templates are created with the OK and Cancel buttons.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting User Interface Fonts

Key Name: HKEY_CURRENT_USER\Software\Microsoft\Developer\Fonts

Value Name: [Normal][Small][Fixed]

Data Type: REG_STRING

Data: *font-name,size*[pt]

This key determines the fonts used by various user interface elements of the system, where:

- *font-name* specifies the actual font name.
- *size* defines the font size in pixels or points.
- [pt] indicates point values. If this field is blank, pixel values are assumed.

The Normal font is used by the status bar, dialog boxes, and browse windows. The Small font is used by toolbars and other docking windows for their captions. The Fixed font is used by the hexadecimal (raw data) editor.

The default fonts listed below are for the U.S. product running on a single-byte character-set system. They are built into the product and do not normally appear in the Registry.

Normal:REG_STRING:MS Sans Serif,8pt

Small:REG_STRING:SmallFonts,-9

Fixed:REG_STRING:Courier,14

Negative numbers specify the character height, and positive numbers specify the cell height. A font's character height is the cell height minus any internal leading.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Setting the Default Magnification Factor

Key Name: HKEY_CURRENT_USER\Software\Microsoft\Developer\Graphics Editor

Value Name: DefaultZoom

Data Type: REG_DWORD

Data: *range*

Sets the default value for the ratio of magnified and actual-size views in the graphics editor window, where:

- *range* is 2 through 10 (if this entry does not appear in the Registry, the default value of 6 is used).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Describing Mouse Pointer Devices

Key Name: HKEY_CURRENT_USER\Software\Microsoft\Developer\Mouse Pointer Devices

Value Name: *device-name*

Data Type: REG_SZ

Data: *number-of-colors,width,height*

Specifies the names of display devices and the attributes of their corresponding mouse pointer images, where:

- *device-name* is the name of the new mouse pointer device that appears in the New Device Image dialog box when you add a mouse pointer image.
- *number-of-colors* specifies the number of colors supported by the device. The number of colors entry must be 2 or 16.
- *width* is the image width in pixels.
- *height* is the image height in pixels.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Describing Icon Devices

Key Name: HKEY_CURRENT_USER\Software\Microsoft\Developer\Icon Devices

Value Name: *device-name*

Data Type: REG_SZ

Data: *number-of-colors,width,height*

Specifies the names of display devices and the attributes of their corresponding icon images, where:

- *device-name* is the name of the new icon device that appears in the New Device Image dialog box when you add an icon image.
- *number-of-colors* specifies the number of colors supported by the device. The number of colors entry must be 2 or 16.
- *width* is the image width in pixels.
- *height* is the image height in pixels.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


JVIEW Reference

JVIEW.EXE (JVIEW) is a tool used to execute Java applications either from inside Developer Studio or from the command line.

JVIEW provides an environment in which your application can run. It supports both debug and retail versions of your application.

JVIEW currently does not support pure Java applets. If your Java project has a **main method**, then JVIEW will execute it. This includes all Java applications, and Java applets generated by the Applet Wizard.

For an alphabetic reference to the JVIEW options see [Reference to JVIEW Command-Line Options](#).

For more information, see:

- [Description of JVIEW Syntax](#)
- [Using JVIEW.EXE](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Description of JVIEW Syntax

The JVIEW.EXE (JVIEW) command line uses the following syntax:

JVIEW [*options*] <*classname*> [*arguments*]

The following table describes input to the JVIEW command.

En	Meaning
try	
<i>opt</i>	One or more
<i>ion</i>	JVIEW options.
<i>s</i>	See
	Reference to JVIEW Command-Line Options
	for more information.
<i>cla</i>	The name of the
<i>ssn</i>	.CLASS file to
<i>am</i>	execute. Do not
<i>e</i>	include the .CLASS extension to this filename. For example, use HelloWorldAp
	p and not HelloWorldAp
	p.class.
<i>arg</i>	Command-line
<i>um</i>	arguments to be
<i>ent</i>	passed to
<i>s</i>	the .CLASS file supplied in
	<i>classname</i> .

Note Any options that you wish to supply to JVIEW must be supplied before the name of the .CLASS file or they will be interpreted as command-line arguments to the .CLASS file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Using JVIEW.EXE

You can specify JVIEW.EXE (JVIEW) options on the command line or in Developer Studio. If you are executing your Java application from within Developer Studio, specify the JVIEW options you wish to use in the Debug pane of the Project Setting dialog box. Choose Stand-alone Interpreter in the drop-down list box. Enter the JVIEW options you wish to use in the Stand-alone interpreter arguments edit field.

If you are executing your Java application from the command line, enter the options you wish to supply to JVIEW before the name of your .CLASS file on the command line.

Note JVIEW will only run Java applications. To use JVIEW for executing your application, your class must contain a main() method. If your program is an applet, run it from a browser.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Reference to JVIEW Command-Line Options

This topic is an alphabetic reference to all the JVIEW command-line options. These options are listed in Table 18.1.

If a command-line option can take one or more arguments, its syntax is shown under a Syntax heading before its description. Click any option in the following table for information on the option.

Table 18.1 JVIEW Options

[/cp](#)

[/cp:a](#)

[/cp:p](#)

The options listed in Table 18.1 may be typed into the Debug tab of the Project Settings dialog box. These options can also be used from the command line.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/cp -- JVIEW Option

Syntax

/cp classpath

Use the /cp option to set the CLASSPATH environment variable for the current compilation. Using this option specifies the path where the JVC or JVIEW command-line tools can find system and user-defined classes. The Java interpreter uses a platform-dependent default location and the CLASSPATH environment variable to find system classes. See CLASSPATH Environment Variable for more information. The directories in the class path are separated by semicolons on a Microsoft Windows system; by colons on Unix.

Example

For example, on Windows NT, the class path might be:

```
JVIEW /cp x:.;x:\java\classes
```

In this example JVIEW searches in and beneath the directories on the path for system and user-defined classes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/cp:a -- JVIEW Option

Syntax

`/cp:a path`

This option appends the *path* entered to the end of the *classpath* environment variable and inserts a semicolon between them.

For a full description of the CLASSPATH environment variable, see CLASSPATH Environment Variable.

Note The value for the class path may come from the CLASSPATH environment variable or from the /cp option.

```
JVIEW /cp:a myproj
```

When multiple /cp:a switches are entered, the paths are concatenated.

Example

The following command concatenates the directories, myproj1 and myproj2, and appends the resulting path to the end of the existing classpath:

```
JVIEW /cp:a myproj1 /cp:a myproj2
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


/cp:p -- JVIEW Option

Syntax

/cp:p *path*

This option prepends the *path* entered to the *class path* and inserts a semicolon between them. When multiple /cp:p switches are entered, the paths are concatenated.

Note The value for the class path may come from the CLASSPATH environment variable or from the /cp option. See CLASSPATH Environment Variable for more information.

```
JVIEW /cp:p myproj
```

When multiple /cp:p switches are entered, the paths are concatenated.

Example

The following command concatenates the directories, myproj1 and myproj2, and prepends the resulting path to the existing class path:

```
JVIEW /cp:p myproj1/cp:p myproj2
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ZoomIn Reference

You can use the ZoomIn utility (ZOOMIN.EXE) to capture and enlarge an area of the Windows desktop.

{ewl msdncd, EWGraphic, jug0k 0 /a "build.bmp"} To use ZoomIn

- 1 Double-click the ZoomIn icon in the Microsoft Visual J++ 1.0 folder or program group.

Note You may need to add the ZoomIn icon to the Microsoft Visual J++ 1.0 folder or program group.

- 2 Click within the ZoomIn window's client area and drag the rectangle over the target area you want to enlarge.

This target area can be anywhere in the Windows graphical desktop area. The area over which you center the ZoomIn rectangle appears in the ZoomIn window's client area.

- 3 Release the mouse button when the desired target area is visible in the ZoomIn window.

To enlarge the image, use the scroll bar to scroll down. To reduce the image to its original size, use the scroll bar to scroll up. Each successive click on the scroll bar enlarges or shrinks the image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ZoomIn Menus

Edit Menu

You can use the Edit menu to copy to the Clipboard and update the ZoomIn window image.

Menu Command	Description
Copy	Copies the contents of the ZoomIn window to the Clipboard.
Refresh	Updates the image in the ZoomIn window. This update is visible only if the Windows desktop target area has changed since it was last captured by ZoomIn.

Options Menu

You can use the Options menu to specify the update interval.

Menu Command	Description
Refresh Rate	Enables the automatic update of the ZoomIn window. This dialog box also allows you to specify, in increments of one-tenth of a second, the automatic update

interval.

Help Menu

You can use the Help menu to display information about ZoomIn.

Menu Command	Description
About	Displays copyright and version information about ZoomIn.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

WinDiff Reference

The WinDiff utility (WINDIFF.EXE) graphically compares the contents of two files or two directories. With WinDiff, you can compare and modify the contents of files and directories using a graphical Windows interface.

{ewl msdncd, EWGraphic, jug0l 0 /a "build.bmp"} To start WinDiff

- Double-click the WinDiff icon in the Microsoft Visual J++ 1.0 folder or program group.
—or—
- Use the WinDiff command line.

Note You may need to add the WinDiff icon to the Microsoft Visual J++ 1.0 folder or program group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

WinDiff Command Line

The full WinDiff command-line syntax is:

WINDIFF *path1* [*path2*] [-s [*options*] *savefile*]

Parameters

path1 Compares files in *path1* with files in current directory.

path1 path2 Compares files in *path1* with files in *path2*.

options Can be any combination of the following options:

- /s: Compares files that are the same in both paths.
- /l: Compares only files in the first (left) path.
- /r: Compares only files in the second (right) path.
- /d: Compares two different files in both paths.

savefile Name of text file where comparison results are written.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Expand/Outline Button

You can display the filenames or the expanded contents of the selected files. When you choose the Expand button, the contents of the files are expanded and the button label then changes to Outline. When you choose the Outline button, only the filenames are displayed. Files with the same name but different contents are displayed in red text. Identical files are displayed in black text.

{ewl msdncd, EWGraphic, jug3l 0 /a "build.bmp"} To display the expanded contents of the files

- 1 Select the files you want to compare.
- 2 From the Expand menu, choose Both Files.
- 3 From the View menu, choose Expand.

—or—

Choose the Expand button.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

WinDiff Colors

File contents are displayed in three background colors.

Back grou nd Colo r	Description
Red	Indicates different text from the first (left) file.
Yello w	Indicates different text from the second (right) file.
White	Indicates identical text from both files.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

WinDiff Menus

File Menu

You can use the File menu to define selection, naming, and printing options.

Menu Command	Description
Compare Files	Displays the File Open dialog box , in which you can enter the names of two files to compare.
Compare Directories	Displays the Select Directories dialog box , in which you can enter the names of two directories to compare.
Abort	Terminates a file-scanning operation. This menu selection is unavailable until a scanning operation is initiated.
Save File List	Displays the Save File List dialog box , in which you can specify the output file where the comparison results are to be written.
Copy Files	Displays the Copy Files dialog box ,

in which you
can specify
files to be
copied from
one directory
to another.

Print Prints the
comparison
results.

Exit Terminates
WinDiff.

Edit Menu

You can use the Edit menu to designate the text editor you want to use and to specify which files to display for editing.

Men u Com man d	Description
-----------------------------	-------------

Edit Left File	Displays the contents of the first (left) file using the default Notepad editor.
----------------------	--

Edit Right File	Displays the contents of the second (right) file using the default Notepad editor.
-----------------------	---

Edit Com posit e File	Displays both files using the default Notepad editor.
--------------------------------	--

Set Edito r	Displays a WinDiff dialog box , in which you can specify the editor to be used for the preceding operations. By default,
-------------------	--

Notepad is used.

View Menu

You can use the View menu to compare both the content and graphical representation of two files.

Menu Command	Description
Outline	Displays only the list of filenames (equivalent to the Outline button).
Expand	Displays comparison of the contents of selected files (equivalent to the Expand button).
Picture	Displays a graphical representation of the contents of the two files.
Previous Change	Goes directly to previous area of the file that was changed (if any).
Next Change	Goes directly to next area of the file that was changed (if any).

Expand Menu

You can use the Expand menu to display changed lines in the selected file. You can also turn off the display of line numbers.

Menu Command	Description
--------------	-------------

d

Left File Only	Expands only the first (left) file, with changed lines colored appropriately.
Right File Only	Expands only the second (right) file, with changed lines colored appropriately.
Both Files	Expands both files, with changed lines colored appropriately.
Left Line Num bers	Displays line numbers for the first (left) file.
Right Line Num bers	Displays line numbers for the second (right) file.
No Line Num bers	Turns off the line number display.

Options Menu

You can use the Options menu to specify file comparison criteria.

Men u Com man d	Description
-----------------------------	-------------

Ignor e Blank s	Ignores blank spaces in the expanded view , so that lines differing only in the amount of white space are shown as identical.
--------------------------	---

Mono Displays
Color differences in
s black and
white only.

Show In outline
Identi [view](#),
cal displays files
Files that are
identical.

Show In outline
Left- [view](#),
Only displays files
Files that appear
only in the
first (left)
path.

Show In outline
Right [view](#),
-Only displays files
Files that appear
only in the
second
(right) path.

Show In outline
Differ [view](#),
ent displays files
Files that are in
both paths,
but are
different.

Mark Menu

You can use the Mark menu to mark comparison results.

Men u Com man d	Description
Mark File	Marks selected comparison results.
Mark Patte rn	Displays the Mark Files dialog box , in which you can specify the file marking pattern.
Hide	Hides all

Mark ed Files	marked files.
Toggl e Mark ed State	Reverses the marked status of marked and unmarked files.

Help Menu

You can use the Help menu to display information about WinDiff.

Men u Com man d	Description
About	Displays copyright and version information about WinDiff.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Java Language Specification

Version 1.0

James Gosling
Bill Joy
Guy Steele

*"When I use a word," Humpty Dumpty said,
in rather a scornful tone, "it means just what I
choose it to mean--neither more nor less."
"The question is," said Alice, "whether you
can make words mean so many different things."
"The question is," said Humpty Dumpty,
"which is to be master that's all."
--Lewis Carroll, *Through the Looking Glass**

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Copyright

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The release described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Sun Microsystems, Inc. (SUN) hereby grants to you a fully-paid, nonexclusive, nontransferable, perpetual, worldwide limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this specification. This license allows and is limited to the creation and distribution of clean room implementations of this specification that (i) include a complete implementation of the current version of this specification without subsetting or supersetting, (ii) implement all the interfaces and functionality of the standard `java.*` packages as defined by SUN, without subsetting or supersetting, (iii) do not add any additional packages, classes or methods to the `java.*` packages (iv) pass all test suites relating to the most recent published version of this specification that are available from SUN six (6) months prior to any beta release of the clean room implementation or upgrade thereto, (v) do not derive from SUN source code or binary materials, and (vi) do not include any SUN binary materials without an appropriate and separate license from SUN.

Sun, Sun Microsystems, Sun Microsystems Computer Corporation, the Sun logo, the Sun Microsystems Computer Corporation logo, Java, JavaSoft, JavaScript and HotJava are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX® is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Apple and Dylan are trademarks of Apple Computer, Inc. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Credits and permissions for quoted material appear in a separate section on page 823.

Text printed on recycled and acid-free paper

ISBN 0-201-63451-1

1 2 3 4 5 6 7 8 9-MA-99989796

First printing, August 1996

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Series Foreword

The Java Series books provide definitive reference documentation for Java programmers and end users. They are written by members of the Java team and published under the auspices of JavaSoft, a Sun Microsystems business. The World-Wide-Web allows Java documentation to be made available over the Internet, either by downloading or as hypertext. Nevertheless, the world-wide interest in Java technology led us to write and publish these books to supplement all of the documentation at our Web site

To learn the latest about the Java Platform and Environment or download the latest Java release, visit our World Wide Web site at <http://java.sun.com>. For updated information about the Java Series, including sample code, errata, and previews of forthcoming books, visit <http://java.sun.com/Series>.

We would like to thank the Corporate and Professional Publishing Group at Addison-Wesley for their partnership in putting together the Series. Our editor Mike Hendrickson and his team have done a superb job of navigating us through the world of publishing. Within Sun Microsystems, the support of James Gosling, Jon Kannegaard, and Bill Joy ensured that this series would have the resources it needed to be successful. In addition to the tremendous effort by individual authors, many members of the JavaSoft team have contributed behind the scenes to bring the highest level of quality and engineering to the books in the Series. A personal note of thanks to my children Christopher and James for putting a positive spin on the many trips to my office during the development of the Series.

Lisa Friendly
Series Editor

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Preface

Java was originally called Oak, and designed for use in embedded consumer-electronic applications by James Gosling. After several years of experience with the language, and significant contributions by Ed Frank, Patrick Naughton, Jonathan Payne, and Chris Warth it was retargeted to the Internet, renamed Java, and substantially revised to be the language specified here. The final form of the language was defined by James Gosling, Bill Joy, Guy Steele, Richard Tuck, Frank Yellin, and Arthur van Hoff, with help from Graham Hamilton, Tim Lindholm and many other friends and colleagues.

Java is a general-purpose concurrent class-based object-oriented programming language, specifically designed to have as few implementation dependencies as possible. Java allows application developers to write a program once and then be able to run it everywhere on the Internet.

This book attempts a complete specification of the syntax and semantics of the Java language and the core packages `java.lang`, `java.io`, and `java.util` of its Application Programming Interface. We intend that the behavior of every language construct is specified here, so that all implementations of Java will accept the same programs. Except for timing dependencies or other non-determinisms and given sufficient time and sufficient memory space, a Java program should compute the same result on all machines and in all implementations.

We believe that Java is a mature language, ready for widespread use. Nevertheless, we expect some evolution of the language in the years to come. We intend to manage this evolution in a way that is completely compatible with existing applications. To do this, we intend to make relatively few new versions of the language, and to distinguish each new version with a different filename extension. Java compilers and systems will be able to support the several versions simultaneously, with complete compatibility.

Much research and experimentation with Java is already underway. We encourage this work, and will continue to cooperate with external groups to explore improvements to Java. For example, we have already received several interesting proposals for parameterized types. In technically difficult areas, near the state of the art, this kind of research collaboration is essential.

We acknowledge and thank the many people who have contributed to this book through their excellent feedback, assistance and encouragement:

Particularly thorough, careful, and thoughtful reviews of drafts were provided by Tom Cargill, Peter Deutsch, Paul Hilfinger, Masayuki Ida, David Moon, Steven Muchnick, Charles L. Perkins, Chris Van Wyk, Steve Vinoski, Philip Wadler, Daniel Weinreb, and Kenneth Zadeck. We are very grateful for their extraordinary volunteer efforts.

We are also grateful for reviews, questions, comments, and suggestions from Stephen Adams, Bowen Alpern, Glenn Ammons, Leonid Arbutov, Kim Bruce, Edwin Chan, David Chase, Pavel Curtis, Drew Dean, William Dietz, David Dill, Patrick Dussud, Ed Felten, John Giannandrea, John Gilmore, Charles Gust, Warren Harris, Lee Hasiuk, Mike Hendrickson, Mark Hill, Urs Hoelzle, Roger Hoover, Susan Flynn Hummel, Christopher Jang, Mick Jordan, Mukesh Kacker, Peter Kessler, James Larus, Derek Lieber, Bill McKeeman, Steve Naroff, Evi Nemeth, Robert O'Callahan, Dave Papay, Craig Partridge, Scott Pfeffer, Eric Raymond, Jim Roskind, Jim Russell, William Scherlis, Edith Schonberg, Anthony Scian, Matthew Self, Janice Shepherd, Kathy Stark, Barbara Steele, Rob Strom, William Waite, Greg Weeks, and Bob Wilson. (This list was generated semi-automatically from our e-mail records. We apologize if we have omitted anyone.)

The feedback from all these reviewers was invaluable to us in improving the definition of the Java language as well as the form of the presentation in this book. We thank them for their diligence. Any remaining errors in this book—we hope they are few—are our responsibility and not theirs.

We thank Francesca Freedman for assistance with matters of typography and layout. We thank Dan Mills of Adobe Systems Incorporated for assistance in exploring possible choices of typefaces.

Many of our colleagues at Sun Microsystems have helped us in one way or another. Lisa Friendly,

our series editor, managed our relationship with Addison-Wesley. Susan Stambaugh managed the distribution of many hundreds of copies of drafts to reviewers. We received valuable assistance and technical advice from Ben Adida, Ole Agesen, Ken Arnold, Rick Cattell, Asmus Freytag, Norm Hardy, Steve Heller, David Hough, Doug Kramer, Nancy Lee, Marianne Mueller, Akira Tanaka, Greg Tarsy, David Ungar, Jim Waldo, Ann Wollrath, Geoff Wyant, and Derek White. We thank Alan Baratz, David Bowen, Mike Clary, John Doerr, Jon Kannegaard, Eric Schmidt, Bob Sproull, Bert Sutherland, and Scott McNealy for leadership and encouragement.

The on-line Bartleby Library of Columbia University, at URL:

<http://www.cc.columbia.edu/acis/bartleby/>

was invaluable to us during the process of researching and verifying many of the quotations that are scattered throughout this book. Here is one example:

They lard their lean books with the fat of others' works.
--Robert Burton (1576-1640)

We are grateful to those who have toiled on Project Bartleby, for saving us a great deal of effort and reawakening our appreciation for the works of Walt Whitman.

We are thankful for the tools and services we had at our disposal in writing this book: telephones, overnight delivery, desktop workstations, laser printers, photocopiers, text formatting and page layout software, fonts, electronic mail, the World Wide Web, and, of course, the Internet. We live in three different states, scattered across a continent, but collaboration with each other and with our reviewers has seemed almost effortless. Kudos to the thousands of people who have worked over the years to make these excellent tools and services work quickly and reliably.

Mike Hendrickson, Katie Duffy, Simone Payment, and Rosa Aime Gonzlez of Addison-Wesley were very helpful, encouraging, and patient during the long process of bringing this book to print. We also thank the copy editors.

Rosemary Simpson worked hard, on a very tight schedule, to create the index. We got into the act at the last minute, however; blame us and not her for any jokes you may find hidden therein.

Finally, we are grateful to our families and friends. Life has been a bit crazy for us these last few months and we thank these good people for their support.

In their book *The C Programming Language*, Brian Kernighan and Dennis Ritchie said that they felt that the C language "wears well as one's experience with it grows." If you like C, we think you will like Java. We hope that Java, too, wears well for you.

James Gosling
Cupertino, California

Bill Joy
Aspen, Colorado

Guy Steele
Chelmsford, Massachusetts

July, 1996

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Introduction

If I have seen further it is by standing upon the shoulders of Giants.

--Sir Isaac Newton

Java is a general-purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough that many programmers can achieve fluency in the language. Java is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included. Java is intended to be a production language, not a research language, and so, as C. A. R. Hoare suggested in his classic paper on language design, the design of Java has avoided including new and untested features.

Java is strongly typed. This specification clearly distinguishes between the compile-time errors that can and must be detected, and those that occur at run time. Compile time normally consists of translating Java programs into a machine-independent byte-code representation. Run-time activities include loading and linking of the classes needed to execute a program, optional machine code generation and dynamic optimization of the program, and actual program execution.

Java is a relatively high-level language, in that details of the machine representation are not available through the language. It includes automatic storage management, typically using a garbage collector, to avoid the safety problems of explicit deallocation (as in C's `free` or C++'s `delete`). High-performance garbage-collected implementations of Java can have bounded pauses to support systems programming and real-time applications. Java does not include any unsafe constructs, such as array accesses without index checking, since such unsafe constructs would cause a program to behave in an unspecified way.

Java is normally compiled to a bytecoded instruction set and binary format defined in *The Java Virtual Machine Specification* (Addison-Wesley, 1996). Most implementations of Java for general-purpose programming will support the additional packages defined in the series of books under the general title *The Java Application Programming Interface* (Addison-Wesley).

This Java Language Specification is organized as follows:

Chapter 2 describes grammars and the notation used to present the lexical and syntactic grammars for Java.

Chapter 3 describes the lexical structure of Java, which is based on C and C++. Java is written in the Unicode character set. Java supports the writing of Unicode characters on systems that support only ASCII.

Chapter 4 describes Java's types, values, and variables. Java's types are the primitive types and reference types.

The primitive types are defined to be the same on all machines and in all implementations, and are various sizes of two's-complement integers, single- and double-precision IEEE 754 standard floating-point numbers, a `boolean` type, and a Unicode character `char` type. Values of the primitive types do not share state.

Java's reference types are the class types, the interface types, and the array types. The reference types are implemented by dynamically created objects that are either instances of classes or arrays. Many references to each object can exist. All objects (including arrays) support the methods of the standard class `Object`, which is the (single) root of the class hierarchy. A predefined `String` class supports Unicode character strings. Standard classes exist for wrapping primitive values inside of objects.

Variables are typed storage locations. A variable of a primitive type holds a value of that exact primitive type. A variable of a class type can hold a null reference or a reference to an object whose type is that class type or any subclass of that class type. A variable of an interface type can hold a

null reference or a reference to an instance of any class that implements the interface. A variable of an array type can hold a null reference or a reference to an array. A variable of class type `Object` can hold a null reference or a reference to any object, whether class instance or array.

Chapter 5 describes Java's conversions and numeric promotions. Conversions change the compile-time type and, sometimes, the value of an expression. Numeric promotions are used to convert the operands of a numeric operator to a common type where an operation can be performed. There are no loopholes in the language; casts on reference types are checked at run time to ensure type safety.

Chapter 6 describes declarations and names, and how to determine what names mean (denote). Java does not require types or their members to be declared before they are used. Declaration order is significant only for local variables and the order of initializers of fields in a class or interface.

Java provides control over the scope of names and supports limitations on external access to members of packages, classes, and interfaces. This helps in writing large programs by distinguishing the implementation of a type from its users and those who extend it. Standard naming conventions that make for more readable programs are described here.

Chapter 7 describes the structure of a Java program, which is organized into packages similar to the modules of Modula. The members of a package are compilation units and subpackages. Compilation units contain type declarations and can import types from other packages to give them short names. Packages have names in a hierarchical namespace, and the Internet domain name system can be used to form unique package names.

Chapter 8 describes Java's classes. The members of classes are fields (variables) and methods. Class variables exist once per class. Class methods operate without reference to a specific object. Instance variables are dynamically created in objects that are instances of classes. Instance methods are invoked on instances of classes; such instances become the current object `this` during their execution, supporting the object-oriented programming style.

Classes support single implementation inheritance, in which the implementation of each class is derived from that of a single superclass, and ultimately from the class `Object`. Variables of a class type can reference an instance of that class or of any subclass of that class, allowing new types to be used with existing methods, polymorphically.

Classes support concurrent programming with `synchronized` methods. Methods declare the checked exceptions that can arise from their execution, which allows compile-time checking to ensure that exceptional conditions are handled. Objects can declare a `finalize` method that will be invoked before the objects are discarded by the garbage collector, allowing the objects to clean up their state.

For simplicity, Java has neither declaration "headers" separate from the implementation of a class nor separate type and class hierarchies.

Although Java does not include parameterized classes, the semantics of arrays are those of a parameterized class with some syntactic sugar. Like the programming language Beta, Java uses a run-time type check when storing references in arrays to ensure complete type safety.

Chapter 9 describes Java's interface types, which declare a set of abstract methods and constants. Classes that are otherwise unrelated can implement the same interface type. A variable of an interface type can contain a reference to any object that implements the interface. Multiple interface inheritance is supported.

Chapter 10 describes Java arrays. Array accesses include bounds checking. Arrays are dynamically created objects and may be assigned to variables of type `Object`. Java supports arrays of arrays, rather than multidimensional arrays.

Chapter 11 describes Java's exceptions, which are nonresuming and fully integrated with the language semantics and concurrency mechanisms. There are three kinds of exceptions: checked exceptions, run-time exceptions, and errors. The compiler ensures that checked exceptions are

properly handled by requiring that a method or constructor can result in a checked exception only if it declares it. This provides compile-time checking that exception handlers exist, and aids programming in the large. Most user-defined exceptions should be checked exceptions. Invalid operations in the program detected by the Java Virtual Machine result in run-time exceptions, such as `NullPointerException`. Errors result from failures detected by the virtual machine, such as `OutOfMemoryError`. Most simple programs do not try to handle errors.

Chapter 12 describes activities that occur during execution of a Java program. A Java program is normally stored as binary files representing compiled classes and interfaces. These binary files can be loaded into a Java Virtual Machine, linked to other classes and interfaces, and initialized.

After initialization, class methods and class variables may be used. Some classes may be instantiated to create new objects of the class type. Objects that are class instances also contain an instance of each superclass of the class, and object creation involves recursive creation of these superclass instances.

When an object is no longer referenced, it may be reclaimed by the garbage collector. If an object declares a finalizer, the finalizer is executed before the object is reclaimed to give the object a last chance to clean up resources that would not otherwise be released. When a class is no longer needed, it may be unloaded; if a class finalizer is declared, it is given a chance to clean up first. Objects and classes may be finalized on exit of the Java Virtual Machine.

Chapter 13 describes binary compatibility, specifying the impact of changes to types on other types that use the changed types but have not been recompiled. These considerations are of interest to developers of types that are to be widely distributed, in a continuing series of versions, often through the Internet. Good program development environments automatically recompile dependent code whenever a type is changed, so most programmers need not be concerned about these details.

Chapter 14 describes Java's blocks and statements, which are based on C and C++. Java has no `goto`, but includes labeled `break` and `continue` statements. Unlike C, Java requires `boolean` expressions in control-flow statements, and does not convert types to `boolean` implicitly, in the hope of catching more errors at compile time. A `synchronized` statement provides basic object-level monitor locking. A `try` statement can include `catch` and `finally` clauses to protect against non-local control transfers.

Chapter 15 describes Java's expressions. Java fully specifies the (apparent) order of evaluation of expressions, for increased determinism and portability. Overloaded methods and constructors are resolved at compile time by picking the most specific method or constructor from those which are applicable. Java chooses which method or constructor by using the same basic algorithm used in languages with richer dispatching, such as CLOS and Dylan, for the future.

Chapter 16 describes the precise way in which Java ensures that local variables are definitely set before use. While all other variables are automatically initialized to a default value, Java does not automatically initialize local variables in order to avoid masking programming errors.

Chapter 17 describes the semantics of Java threads and locks, which are based on the monitor-based concurrency originally introduced with the Mesa programming language. Java specifies a memory model for shared-memory multiprocessors that supports high-performance implementations.

Chapter 18 describes the facilities for automatically generating documentation from special comments in Java source code.

Chapter 19 presents a LALR(1) syntactic grammar for Java, and describes the differences between this grammar and the expository grammar used in the body of the language specification that precedes it.

Chapters 20 through 22 are the reference manual for the core of the standard Java Application Programming Interface. These packages must be included in all general purpose Java systems.

Chapter 20 describes the package `java.lang`. The types defined in `java.lang` are automatically imported to be available without qualification in all Java programs. They include the primordial class

`Object`, which is a superclass of all other classes; classes such as `Integer` and `Float`, which wrap the primitive types inside objects; exceptions and errors defined by the language and the Java Virtual Machine; `Thread` support; metalinguistic classes such as `Class` and `ClassLoader`; and the class `System`, which abstracts the host system.

Chapter 21 describes the package `java.util`, which defines a few basic utility classes, such as a hashtable class and a pseudo-random number generator.

Chapter 22 describes the package `java.io`, which defines basic input/output facilities, including random access files and streams of values of primitive types.

The book concludes with two indexes: one for the types, methods, and fields defined and described in this specification, and the other a more traditional index.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


1.1 Example Programs

Most of the example programs given in the text are ready to be executed by a Java system and are similar in form to:

```
class Test {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.print(i == 0 ? args[i] : " " + args[i]);  
        System.out.println();  
    }  
}
```

On a Sun workstation, this class, stored in the file `Test.java`, can be compiled and executed by giving the commands:

```
javac Test.java  
java Test Hello, world.
```

producing the output:

```
Hello, world.
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


1.2 References

Apple Computer. *Dylan Reference Manual*. Apple Computer Inc., Cupertino, California. September 29, 1995. See also <http://www.cambridge.apple.com>.

Bobrow, Daniel G., Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon. *Common Lisp Object System Specification*, X3J13 Document 88-002R, June 1988; appears as Chapter 28 of Steele, Guy. *Common Lisp: The Language*, 2nd ed. Digital Press, 1990, ISBN 1-55558-041-6, 770-864.

Ellis, Margaret A., and Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1990, reprinted with corrections October 1992, ISBN 0-201-51459-1.

Harbison, Samuel. *Modula-3*. Prentice Hall, Englewood Cliffs, New Jersey, 1992, ISBN 0-13-596396.

Hoare, C. A. R. *Hints on Programming Language Design*. Stanford University Computer Science Department Technical Report No. CS-73-403, December 1973. Reprinted in SIGACT/SIGPLAN Symposium on Principles of Programming Languages. Association for Computing Machinery, New York, October 1973.

IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1985. Available from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112-5704 USA; 800-854-7179.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*, 2nd ed. Prentice Hall, Englewood Cliffs, New Jersey, 1988, ISBN 0-13-110362-8.

Madsen, Ole Lehrmann, Birger Mller-Pedersen, and Kristen Nygaard. *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, Reading, Massachusetts, 1993, ISBN 0-201-52430.

Mitchell, James G., William Maybury, and Richard Sweet. *The Mesa Programming Language*, Version 5.0. Xerox PARC, Palo Alto, California, CSL 79-3, April 1979.

Stroustrup, Bjarne. *The C++ Programming Language*, 2nd ed. Addison-Wesley, Reading, Massachusetts, 1991, reprinted with corrections January 1994, ISBN 0-201-53992-6.

Unicode Consortium, The. *The Unicode Standard: Worldwide Character Encoding, Version 1.0*. Addison-Wesley, Reading, Massachusetts, Volume 1, 1991, ISBN 0-201-56788-1, and Volume 2, 1992, ISBN 0-201-60845-6. (Version 2, forthcoming, 1996.)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Grammars

Grammar, which knows how to control even kings . . .

--Moliere, Les Femmes Savantes (1672), Act II, scene vi

This chapter describes the context-free grammars used in this specification to define the lexical and syntactic structure of a Java program.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


2.1 Context-Free Grammars

A *context-free grammar* consists of a number of *productions*. Each production has an abstract symbol called a *nonterminal* as its *left-hand side*, and a sequence of one or more nonterminal and *terminal* symbols as its *right-hand side*. For each grammar, the terminal symbols are drawn from a specified *alphabet*.

Starting from a sentence consisting of a single distinguished nonterminal, called the *goal symbol*, a given context-free grammar specifies a *language*, namely, the infinite set of possible sequences of terminal symbols that can result from repeatedly replacing any nonterminal in the sequence with a right-hand side of a production for which the nonterminal is the left-hand side.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

2.2 The Lexical Grammar

A *lexical grammar* for Java is given in §3. This grammar has as its terminal symbols the characters of the Unicode character set. It defines a set of productions, starting from the goal symbol *Input* (§3.5), that describe how sequences of Unicode characters (§3.1) are translated into a sequence of input elements (§3.5).

These input elements, with white space (§3.6) and comments (§3.7) discarded, form the terminal symbols for the syntactic grammar for Java and are called Java *tokens* (§3.5). These tokens are the identifiers (§3.8), keywords (§3.9), literals (§3.10), separators (§3.11), and operators (§3.12) of the Java language.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


2.3 The Syntactic Grammar

The *syntactic grammar* for Java is given in Chapters 4, 6-10, 14, and 15. This grammar has Java tokens defined by the lexical grammar as its terminal symbols. It defines a set of productions, starting from the goal symbol *CompilationUnit* (§7.3), that describe how sequences of tokens can form syntactically correct Java programs.

A LALR(1) version of the syntactic grammar is presented in Chapter 19. The grammar in the body of this specification is very similar to the LALR(1) grammar but more readable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


2.4 Grammar Notation

Terminal symbols are shown in `fixed width` font in the productions of the lexical and syntactic grammars, and throughout this specification whenever the text is directly referring to such a terminal symbol. These are to appear in a program exactly as written.

Nonterminal symbols are shown in *italic* type. The definition of a nonterminal is introduced by the name of the nonterminal being defined followed by a colon. One or more alternative right-hand sides for the nonterminal then follow on succeeding lines. For example, the syntactic definition:

```
IfThenStatement:  
  
    if ( Expression ) Statement
```

states that the nonterminal *IfThenStatement* represents the token `if`, followed by a left parenthesis token, followed by an *Expression*, followed by a right parenthesis token, followed by a *Statement*. As another example, the syntactic definition:

```
ArgumentList:  
  
    Argument  
  
    ArgumentList , Argument
```

states that an *ArgumentList* may represent either a single *Argument* or an *ArgumentList*, followed by a comma, followed by an *Argument*. This definition of *ArgumentList* is *recursive*, that is to say, it is defined in terms of itself. The result is that an *ArgumentList* may contain any positive number of arguments. Such recursive definitions of nonterminals are common.

The subscripted suffix "*opt*", which may appear after a terminal or nonterminal, indicates an *optional symbol*. The alternative containing the optional symbol actually specifies two right-hand sides, one that omits the optional element and one that includes it. This means that:

```
BreakStatement:  
  
    break Identifieropt ;
```

is a convenient abbreviation for:

```
BreakStatement:  
  
    break ;  
  
    break Identifier ;
```


and that:

ForStatement:

for (ForInitopt ; Expressionopt ; ForUpdateopt) Statement

is a convenient abbreviation for:

ForStatement:

for (; Expressionopt ; ForUpdateopt) Statement

for (ForInit ; Expressionopt ; ForUpdateopt) Statement

which in turn is an abbreviation for:

ForStatement:

for (; ; ForUpdateopt) Statement

for (; Expression ; ForUpdateopt) Statement

for (ForInit ; ; ForUpdateopt) Statement

for (ForInit ; Expression ; ForUpdateopt) Statement

which in turn is an abbreviation for:

ForStatement:

for (; ;) Statement

for (; ; ForUpdate) Statement

for (; Expression ;) Statement

for (; Expression ; ForUpdate) Statement

for (ForInit ; ;) Statement

for (ForInit ; ; ForUpdate) Statement

for (ForInit ; Expression ;) Statement

for (ForInit ; Expression ; ForUpdate) Statement

so the nonterminal *ForStatement* actually has eight alternative right-hand sides.

A very long right-hand side may be continued on a second line by substantially indenting this second line, as in:

ConstructorDeclaration:

ConstructorModifiersopt ConstructorDeclarator
Throwsopt ConstructorBody

which defines one right-hand side for the nonterminal *ConstructorDeclaration*. (This right-hand side is an abbreviation for four alternative right-hand sides, because of the two occurrences of "opt".)

When the words "one of" follow the colon in a grammar definition, they signify that each of the terminal symbols on the following line or lines is an alternative definition. For example, the lexical grammar for Java contains the production:

ZeroToThree: one of
0 1 2 3

which is merely a convenient abbreviation for:

ZeroToThree:
0
1
2
3

When an alternative in a lexical production appears to be a token, it represents the sequence of characters that would make up such a token. Thus, the definition:

BooleanLiteral: one of
true false

in a lexical grammar production is shorthand for:

BooleanLiteral:

t r u e

f a l s e

The right-hand side of a lexical production may specify that certain expansions are not permitted by using the phrase "but not" and then indicating the expansions to be excluded, as in the productions for *InputCharacter* (§3.4) and *Identifier* (§3.8):

InputCharacter:

UnicodeInputCharacter but not CR or LF

Identifier:

IdentifierName but not a Keyword or BooleanLiteral or NullLiteral

Finally, a few nonterminal symbols are described by a descriptive phrase in roman type in cases where it would be impractical to list all the alternatives:

RawInputCharacter:

any Unicode character

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Lexical Structure

Lexicographer: A writer of dictionaries, a harmless drudge.

--Samuel Johnson, Dictionary (1755)

This chapter specifies the lexical structure of Java.

Java programs are written in Unicode (§3.1), but lexical translations are provided (§3.2) so that Unicode escapes (§3.3) can be used to include any Unicode character using only ASCII characters. Line terminators are defined (§3.4) to support the different conventions of existing host systems while maintaining consistent line numbers.

The Unicode characters resulting from the lexical translations are reduced to a sequence of input elements (§3.5), which are white space (§3.6), comments (§3.7), and tokens. The tokens are the identifiers (§3.8), keywords (§3.9), literals (§3.10), separators (§3.11), and operators (§3.12) of the Java syntactic grammar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.1 Unicode

Java programs are written using the Unicode character set, version 2.0. Information about this encoding may be found at:

`http://www.unicode.org` and `ftp://unicode.org`

Versions of Java prior to 1.1 used Unicode version 1.1.5 (see *The Unicode Standard: Worldwide Character Encoding* (§1.2) and updates). See §20.5 for a discussion of the differences between Unicode version 1.1.5 and Unicode version 2.0.

Except for comments (§3.7), identifiers, and the contents of character and string literals (§3.10.4, §3.10.5), all input elements (§3.5) in a Java program are formed only from ASCII characters (or Unicode escapes (§3.3) which result in ASCII characters). ASCII (ANSI X3.4) is the American Standard Code for Information Interchange. The first 128 characters of the Unicode character encoding are the ASCII characters.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

3.2 Lexical Translations

A raw Unicode character stream is translated into a sequence of Java tokens, using the following three lexical translation steps, which are applied in turn:

- A translation of Unicode escapes (§3.3) in the raw stream of Unicode characters to the corresponding Unicode character. A Unicode escape of the form `\uxxxx`, where `xxxx` is a hexadecimal value, represents the Unicode character whose encoding is `xxxx`. This translation step allows any Java program to be expressed using only ASCII characters.
- A translation of the Unicode stream resulting from step 1 into a stream of input characters and line terminators (§3.4).
- A translation of the stream of input characters and line terminators resulting from step 2 into a sequence of Java input elements (§3.5) which, after white space (§3.6) and comments (§3.7) are discarded, comprise the tokens (§3.5) that are the terminal symbols of the syntactic grammar (§2.3) for Java.

Java always uses the longest possible translation at each step, even if the result does not ultimately make a correct Java program, while another lexical translation would. Thus the input characters `a--b` are tokenized (§3.5) as `a`, `--`, `b`, which is not part of any grammatically correct Java program, even though the tokenization `a`, `-`, `-`, `b` could be part of a grammatically correct Java program.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.3 Unicode Escapes

Java implementations first recognize *Unicode escapes* in their input, translating the ASCII characters `\u` followed by four hexadecimal digits to the Unicode character with the indicated hexadecimal value, and passing all other characters unchanged. This translation step results in a sequence of Unicode input characters:

UnicodeInputCharacter:

UnicodeEscape

RawInputCharacter

UnicodeEscape:

`\ UnicodeMarker HexDigit HexDigit HexDigit HexDigit`

UnicodeMarker:

`u`

`UnicodeMarker u`

RawInputCharacter:

`any Unicode character`

HexDigit: one of

`0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F`

The `\`, `u`, and hexadecimal digits here are all ASCII characters.

In addition to the processing implied by the grammar, for each raw input character that is a backslash `\`, input processing must consider how many other `\` characters contiguously precede it, separating it from a non-`\` character or the start of the input stream. If this number is even, then the `\` is eligible to begin a Unicode escape; if the number is odd, then the `\` is not eligible to begin a Unicode escape. For example, the raw input `"\\u2297=\u2297"` results in the eleven characters `" \ \ u 2 2 9 7 = {ewc msdn cd, EWGraphic, LNG3g 0 /a "langref.BMP"} "` (`\u2297` is the Unicode encoding of the character `{ewc msdn cd, EWGraphic, LNG3g 1 /a "langref.BMP"}`).

If an eligible `\` is not followed by `u`, then it is treated as a *RawInputCharacter* and remains part of the escaped Unicode stream. If an eligible `\` is followed by `u`, or more than one `u`, and the last `u` is not followed by four hexadecimal digits, then a compile-time error occurs.

The character produced by a Unicode escape does not participate in further Unicode escapes. For example, the raw input `\u005cu005a` results in the six characters `\ u 0 0 5 a`, because `005c` is the Unicode value for `\`. It does not result in the character `z`, which is Unicode character `005a`, because the `\` that resulted from the `\u005c` is not interpreted as the start of a further Unicode escape.

Java specifies a standard way of transforming a Unicode Java program into ASCII that changes a Java program into a form that can be processed by ASCII-based tools. The transformation involves converting any Unicode escapes in the source text of the program to ASCII by adding an extra `u-` for

example, `\uxxxx` becomes `\uuxxxx`-while simultaneously converting non-ASCII characters in the source text to a `\uxxxx` escape containing a single `u`. This transformed version is equally acceptable to a Java compiler and represents the exact same program. The exact Unicode source can later be restored from this ASCII form by converting each escape sequence where multiple `u`'s are present to a sequence of Unicode characters with one fewer `u`, while simultaneously converting each escape sequence with a single `u` to the corresponding single Unicode character.

Java systems should use the `\uxxxx` notation as an output format to display Unicode characters when a suitable font is not available.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.4 Line Terminators

Java implementations next divide the sequence of Unicode input characters into lines by recognizing *line terminators*. This definition of lines determines the line numbers produced by a Java compiler or other Java system component. It also specifies the termination of the `//` form of a comment (§3.7).

LineTerminator:

the ASCII LF character, also known as "newline"

the ASCII CR character, also known as "return"

the ASCII CR character followed by the ASCII LF character

InputCharacter:

UnicodeInputCharacter but not CR or LF

Lines are terminated by the ASCII characters CR, or LF, or CR LF. The two characters CR immediately followed by LF are counted as one line terminator, not two. The result is a sequence of line terminators and input characters, which are the terminal symbols for the third step in the tokenization process.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.5 Input Elements and Tokens

The input characters and line terminators that result from escape processing (§3.3) and then input line recognition (§3.4) are reduced to a sequence of *input elements*. Those input elements that are not white space (§3.6) or comments (§3.7) are *tokens*. The tokens are the terminal symbols of the Java syntactic grammar (§2.3).

This process is specified by the following productions:

Input:

*InputElements*opt *Sub*opt

InputElements:

InputElement

InputElements InputElement

InputElement:

WhiteSpace

Comment

Token

Token:

Identifier

Keyword

Literal

Separator

Operator

Sub:

the ASCII SUB character, also known as "control-Z"

White space (§3.6) and comments (§3.7) can serve to separate tokens that, if adjacent, might be tokenized in another manner. For example, the ASCII characters – and = in the input can form the operator token –= (§3.12) only if there is no intervening white space or comment.

As a special concession for compatibility with certain operating systems, the ASCII SUB character (\u001a, or control-Z) is ignored if it is the last character in the escaped input stream.

Consider two tokens *x* and *y* in the resulting input stream. If *x* precedes *y*, then we say that *x* is *to the left of y* and that *y* is *to the right of x*. For example, in this simple piece of Java code:


```
class Empty {  
}
```

we say that the `}` token is to the right of the `{` token, even though it appears, in this two-dimensional representation on paper, downward and to the left of the `{` token. This convention about the use of the words left and right allows us to speak, for example, of the right-hand operand of a binary operator or of the left-hand side of an assignment.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.6 White Space

White space is defined as the ASCII space, horizontal tab, and form feed characters, as well as line terminators (§3.4).

WhiteSpace:

the ASCII SP character, also known as "space"

the ASCII HT character, also known as "horizontal tab"

the ASCII FF character, also known as "form feed"

LineTerminator

{ewl msdncl.dll, ewcright, /c"Microsoft"}

3.7 Comments

Java defines three kinds of *comments*:

/ text */* *A traditional comment*: all the text from the ASCII characters */** to the ASCII characters **/* is ignored (as in C and C++).

// text *A single-line comment*: all the text from the ASCII characters *//* to the end of the line is ignored (as in C++).

*/** documentation */* *A documentation comment*: the text enclosed by the ASCII characters */*** and **/* can be processed by a separate tool to prepare automatically generated documentation of the following class, interface, constructor, or member (method or field) declaration. See [§18](#) for a full description of how the supplied *documentation* is processed.

These comments are formally specified by the following productions:

Comment:

TraditionalComment

EndOfLineComment

DocumentationComment

TraditionalComment:

*/ * NotStar CommentTail*

EndOfLineComment:

// CharactersInLineopt LineTerminator

DocumentationComment:

*/ * * CommentTailStar*

CommentTail:

** CommentTailStar*

NotStar CommentTail

CommentTailStar:

/

** CommentTailStar*

NotStarNotSlash CommentTail

NotStar:

*InputCharacter but not **

LineTerminator

NotStarNotSlash:

*InputCharacter but not * or /*

LineTerminator

CharactersInLine:

InputCharacter

CharactersInLine InputCharacter

These productions imply all of the following properties:

- Comments do not nest.
- `/*` and `*/` have no special meaning in comments that begin with `//`.
- `//` has no special meaning in comments that begin with `/*` or `/**`.

As a result, the text:

```
/* this comment /* // /** ends here: */
```

is a single complete comment.

The lexical grammar implies that comments do not occur within character literals ([§3.10.4](#)) or string literals ([§3.10.5](#)).

Note that `/**/` is considered to be a documentation comment, while `/* */` (with a space between the asterisks) is a traditional comment.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.8 Identifiers

An *identifier* is an unlimited-length sequence of *Java letters* and *Java digits*, the first of which must be a Java letter. An identifier cannot have the same spelling (Unicode character sequence) as a keyword (§3.9), Boolean literal (§3.10.3), or the null literal (§3.10.7).

Identifier:

IdentifierChars but not a Keyword or BooleanLiteral or NullLiteral

IdentifierChars:

JavaLetter

IdentifierChars JavaLetterOrDigit

JavaLetter:

any Unicode character that is a Java letter (see below)

JavaLetterOrDigit:

any Unicode character that is a Java letter-or-digit (see below)

Letters and digits may be drawn from the entire Unicode character set, which supports most writing scripts in use in the world today, including the large sets for Chinese, Japanese, and Korean. This allows Java programmers to use identifiers in their programs that are written in their native languages.

A Java letter is a character for which the method `Character.isJavaLetter` (§20.5.17) returns `true`. A Java letter-or-digit is a character for which the method `Character.isJavaLetterOrDigit` (§20.5.18) returns `true`.

The Java letters include uppercase and lowercase ASCII Latin letters A-Z (\u0041-\u005a), and a-z (\u0061-\u007a), and, for historical reasons, the ASCII underscore (`_`, or \u005f) and dollar sign (`$`, or \u0024). The `$` character should be used only in mechanically generated Java code or, rarely, to access preexisting names on legacy systems.

The Java digits include the ASCII digits 0–9 (\u0030-\u0039).

Two identifiers are the same only if they are identical, that is, have the same Unicode character for each letter or digit.

Identifiers that have the same external appearance may yet be different. For example, the identifiers consisting of the single letters LATIN CAPITAL LETTER A (A, \u0041), LATIN SMALL LETTER A (a, \u0061), GREEK CAPITAL LETTER ALPHA (Α, \u0391), and CYRILLIC SMALL LETTER A (а, \u0430) are all different.

Unicode composite characters are different from the decomposed characters. For example, a LATIN CAPITAL LETTER A ACUTE (Á, \u00c1) could be considered to be the same as a LATIN CAPITAL LETTER A (A, \u0041) immediately followed by a NON-SPACING ACUTE (´, \u0301) when sorting, but these are different in Java identifiers. See *The Unicode Standard*, Volume 1, pages 412ff for details about decomposition, and see pages 626-627 of that work for details about sorting.

Examples of identifiers are:

```
String          i3          {ewc msdncd, EWGraphic, LNG8g
0 /a "langref.BMP"}{ewc msdncd, EWGraphic, LNG8g 1 /a "langref.BMP"}{ewc
msdncd, EWGraphic, LNG8g 2 /a "langref.BMP"}{ewc msdncd, EWGraphic, LNG8g
3 /a "langref.BMP"}{ewc msdncd, EWGraphic, LNG8g 4 /a "langref.BMP"}
MAX_VALUE          isLetterOrDigit
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.9 Keywords

The following character sequences, formed from ASCII letters, are reserved for use as *keywords* and cannot be used as identifiers (§3.8):

Keyword: one of

```
abstract    default    if      private    throw
boolean     do      implements protected throws
break double      import      public     transient
byte  else  instanceof  return      try
case  extends      int    short void
catch final interface  static      volatile
char  finally      long  super while
class float native      switch
const for   new    synchronized
continue  goto  package    this
```

The keywords `const` and `goto` are reserved by Java, even though they are not currently used in Java. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in Java programs.

While `true` and `false` might appear to be keywords, they are technically Boolean literals (§3.10.3). Similarly, while `null` might appear to be a keyword, it is technically the null literal (§3.10.7).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10 Literals

A *literal* is the source code representation of a value of a primitive type ([§4.2](#)), the `String` type ([§4.3.3](#), [§20.12](#)), or the null type ([§4.1](#)):

Literal:

IntegerLiteral

FloatingPointLiteral

BooleanLiteral

CharacterLiteral

StringLiteral

NullLiteral

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10.1 Integer Literals

See [§4.2.1](#) for a general discussion of the integer types and values.

An *integer literal* may be expressed in decimal (base 10), hexadecimal (base 16), or octal (base 8):

IntegerLiteral:

DecimalIntegerLiteral

HexIntegerLiteral

OctalIntegerLiteral

DecimalIntegerLiteral:

DecimalNumeral IntegerTypeSuffixopt

HexIntegerLiteral:

HexNumeral IntegerTypeSuffixopt

OctalIntegerLiteral:

OctalNumeral IntegerTypeSuffixopt

IntegerTypeSuffix: one of

l L

An integer literal is of type `long` if it is suffixed with an ASCII letter `L` or `l` (ell); otherwise it is of type `int` ([§4.2.1](#)). The suffix `L` is preferred, because the letter `l` (ell) is often hard to distinguish from the digit `1` (one).

A decimal numeral is either the single ASCII character `0`, representing the integer zero, or consists of an ASCII digit from `1` to `9`, optionally followed by one or more ASCII digits from `0` to `9`, representing a positive integer:

DecimalNumeral:

0

NonZeroDigit Digitsopt

Digits:

Digit

Digits Digit

Digit:

0

NonZeroDigit

NonZeroDigit: one of

1 2 3 4 5 6 7 8 9

A hexadecimal numeral consists of the leading ASCII characters `0x` or `0X` followed by one or more ASCII hexadecimal digits and can represent a positive, zero, or negative integer. Hexadecimal digits with values 10 through 15 are represented by the ASCII letters `a` through `f` or `A` through `F`, respectively; each letter used as a hexadecimal digit may be uppercase or lowercase.

HexNumeral:

0 x HexDigit

0 X HexDigit

HexNumeral HexDigit

The following production from §3.3 is repeated here for clarity:

HexDigit: one of

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

An octal numeral consists of an ASCII digit `0` followed by one or more of the ASCII digits `0` through `7` and can represent a positive, zero, or negative integer.

OctalNumeral:

0 OctalDigit

OctalNumeral OctalDigit

OctalDigit: one of

0 1 2 3 4 5 6 7

Note that octal numerals are always consist of two or more digits; `0` is always considered to be a decimal numeral-not that it matters much in practice, for the numerals `0`, `00`, and `0x0` all represent exactly the same integer value.

The largest decimal literal of type `int` is 2147483648 ({ewc msdncd, EWGraphic, LNG11g 0 /a "langrefANC.BMP"}). All decimal literals from 0 to 2147483647 may appear anywhere an `int` literal may appear, but the literal 2147483648 may appear only as the operand of the unary negation operator `-`.

The largest positive hexadecimal and octal literals of type `int` are 0x7fffffff and 017777777777, respectively, which equal 2147483647 ({ewc msdncd, EWGraphic, LNG11g 1 /a "langrefANC1.BMP"}). The most negative hexadecimal and octal literals of type `int` are 0x80000000 and 020000000000, respectively, each of which represents the decimal value -2147483648 ({ewc msdncd, EWGraphic, LNG11g 2 /a "langrefANC2.BMP"}). The hexadecimal and octal literals 0xffffffff and 037777777777, respectively, represent the decimal value -1.

See also `Integer.MIN_VALUE` (§20.7.1) and `Integer.MAX_VALUE` (§20.7.2).

A compile-time error occurs if a decimal literal of type `int` is larger than 2147483648 ({ewc msdncd, EWGraphic, LNG11g 3 /a "langrefANC3.BMP"}), or if the literal 2147483648 appears anywhere other than as the operand of the unary `-` operator, or if a hexadecimal or octal `int` literal does not fit in 32 bits.

Examples of `int` literals:

0	2	0372	0xDadaCafe
1996		0x00FF00FF	

The largest decimal literal of type `long` is 9223372036854775808L ({ewc msdncd, EWGraphic, LNG11g 4 /a "langrefANC4.BMP"}). All decimal literals from 0L to 9223372036854775807L may appear anywhere a `long` literal may appear, but the literal 9223372036854775808L may appear only as the operand of the unary negation operator `-`.

The largest positive hexadecimal and octal literals of type `long` are 0x7fffffffffffffffL and 07777777777777777777L, respectively, which equal 9223372036854775807L ({ewc msdncd, EWGraphic, LNG11g 5 /a "langrefANC5.BMP"}). The literals 0x8000000000000000L and 0100000000000000000000L are the most negative `long` hexadecimal and octal literals, respectively. Each has the decimal value -9223372036854775808L ({ewc msdncd, EWGraphic, LNG11g 6 /a "langrefANC6.BMP"}). The hexadecimal and octal literals 0xffffffffffffffffL and 01777777777777777777L, respectively, represent the decimal value -1L.

See also `Long.MIN_VALUE` (§20.8.1) and `Long.MAX_VALUE` (§20.8.2).

A compile-time error occurs if a decimal literal of type `long` is larger than 9223372036854775808L ({ewc msdncd, EWGraphic, LNG11g 7 /a "langrefANC7.BMP"}), or if the literal 9223372036854775808L appears anywhere other than as the operand of the unary `-` operator, or if a hexadecimal or octal `long` literal does not fit in 64 bits.

Examples of `long` literals:

0L	0777L	0x100000000L
2147483648L		0xC0B0L


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10.2 Floating-Point Literals

See [§4.2.3](#) for a general discussion of the floating-point types and values.

A *floating-point literal* has the following parts: a whole-number part, a decimal point (represented by an ASCII period character), a fractional part, an exponent, and a type suffix. The exponent, if present, is indicated by the ASCII letter `e` or `E` followed by an optionally signed integer.

At least one digit, in either the whole number or the fraction part, and either a decimal point, an exponent, or a float type suffix are required. All other parts are optional.

A floating-point literal is of type `float` if it is suffixed with an ASCII letter `F` or `f`; otherwise its type is `double` and it can optionally be suffixed with an ASCII letter `D` or `d`.

FloatingPointLiteral:

Digits . Digitsopt ExponentPartopt FloatTypeSuffixopt

. Digits ExponentPartopt FloatTypeSuffixopt

Digits ExponentPart FloatTypeSuffixopt

Digits ExponentPartopt FloatTypeSuffix

ExponentPart:

ExponentIndicator SignedInteger

ExponentIndicator: one of

e E

SignedInteger:

Signopt Digits

Sign: one of

+ -

FloatTypeSuffix: one of

f F d D

The Java types `float` and `double` are IEEE 754 32-bit single-precision and 64-bit double-precision binary floating-point values, respectively.

The details of proper input conversion from a Unicode string representation of a floating-point number to the internal IEEE 754 binary floating-point representation are described for the methods `valueOf` of class `Float` ([§20.9.17](#)) and class `Double` ([§20.10.16](#)) of the package `java.lang`.

The largest positive finite `float` literal is `3.40282347e+38f`. The smallest positive finite nonzero literal of type `float` is `1.40239846e-45f`. The largest positive finite `double` literal is `1.79769313486231570e+308`. The smallest positive finite nonzero literal of type `double` is

4.94065645841246544e-324.

See `Float.MIN_VALUE` (§20.9.1) and `Float.MAX_VALUE` (§20.9.2); see also `Double.MIN_VALUE` (§20.10.1) and `Double.MAX_VALUE` (§20.10.2).

A compile-time error occurs if a nonzero floating-point literal is too large, so that on rounded conversion to its internal representation it becomes an IEEE 754 infinity. A Java program can represent infinities without producing a compile-time error by using constant expressions such as `1f/0f` or `-1d/0d` or by using the predefined constants `POSITIVE_INFINITY` and `NEGATIVE_INFINITY` of the classes `Float` (§20.9) and `Double` (§20.10).

A compile-time error occurs if a nonzero floating-point literal is too small, so that, on rounded conversion to its internal representation, it becomes a zero. A compile-time error does not occur if a nonzero floating-point literal has a small value that, on rounded conversion to its internal representation, becomes a nonzero denormalized number.

Predefined constants representing Not-a-Number values are defined in the classes `Float` and `Double` as `Float.NaN` (§20.9.5) and `Double.NaN` (§20.10.5).

Examples of `float` literals:

```
1e1f          2.f          .3f          0f          3.14f
6.022137e+23f
```

Examples of `double` literals:

```
1e1          2.          .3          0.0          3.14
1e-9d        1e137
```

There is no provision for expressing floating-point literals in other than decimal radix. However, method `intBitsToFloat` (§20.9.23) of class `Float` and method `longBitsToDouble` (§20.10.22) of class `Double` provide a way to express floating-point values in terms of hexadecimal or octal integer literals. For example, the value of:

```
Double.longBitsToDouble(0x400921FB54442D18L)
```

is equal to the value of `Math.PI` (§20.11.2).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10.3 Boolean Literals

The `boolean` type has two values, represented by the literals `true` and `false`, formed from ASCII letters.

A *boolean literal* is always of type `boolean`.

BooleanLiteral: one of

true false

{`ewl msdncd.dll, ewcright, /c"Microsoft"`}

3.10.4 Character Literals

A *character literal* is expressed as a character or an escape sequence, enclosed in ASCII single quotes. (The single-quote, or apostrophe, character is `\u0027`.)

A character literal is always of type `char`.

CharacterLiteral:

`' SingleCharacter '`

`' EscapeSequence '`

SingleCharacter:

`InputCharacter but not ' or \`

The escape sequences are described in [§3.10.6](#).

As specified in [§3.4](#), the characters CR and LF are never an *InputCharacter*; they are recognized as constituting a *LineTerminator*.

It is a compile-time error for the character following the *SingleCharacter* or *EscapeSequence* to be other than a `'`.

It is a compile-time error for a line terminator to appear after the opening `'` and before the closing `'`.

The following are examples of `char` literals:

```
'a'
'%'
'\t'
'\'
'\u03a9'
'\uFFFF'
'\177'
'{ewc msdn cd, EWGraphic, LNG14g 0 /a "langref.BMP"}'
'{ewc msdn cd, EWGraphic, LNG14g 1 /a "langref.BMP"}'
```

Because Unicode escapes are processed very early, it is not correct to write `'\u000a'` for a character literal whose value is linefeed (LF); the Unicode escape `\u000a` is transformed into an actual linefeed in translation step 1 ([§3.3](#)) and the linefeed becomes a *LineTerminator* in step 2 ([§3.4](#)), and so the character literal is not valid in step 3. Instead, one should use the escape sequence `'\n'` ([§3.10.6](#)). Similarly, it is not correct to write `'\u000d'` for a character literal whose value is carriage return (CR). Instead, use `'\r'`.

In C and C++, a character literal may contain representations of more than one character, but the value of such a character literal is implementation-defined. In Java, a character literal always represents exactly one character.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10.5 String Literals

A *string literal* consists of zero or more characters enclosed in double quotes. Each character may be represented by an escape sequence.

A string literal is always of type `String` (§4.3.3, §20.12). A string literal always refers to the same instance (§4.3.1) of class `String`.

StringLiteral:

" StringCharactersopt "

StringCharacters:

StringCharacter

StringCharacters StringCharacter

StringCharacter:

*InputCharacter but not " or *

EscapeSequence

The escape sequences are described in §3.10.6.

As specified in §3.4, neither of the characters CR and LF is ever considered to be an *InputCharacter*; each is recognized as constituting a *LineTerminator*.

It is a compile-time error for a line terminator to appear after the opening `"` and before the closing `"`. A long string literal can always be broken up into shorter pieces and written as a (possibly parenthesized) expression using the string concatenation operator `+` (§15.17.1).

The following are examples of string literals:

```
""                // the empty string
"\\"             // a string containing " alone
"This is a string" // a string
containing 16 characters

"This is a " +    // actually a string-
valued constant expression,
    "two-line string" //          formed
from two string literals
```

Because Unicode escapes are processed very early, it is not correct to write `"\u000a"` for a string literal containing a single linefeed (LF); the Unicode escape `\u000a` is transformed into an actual linefeed in translation step 1 (§3.3) and the linefeed becomes a *LineTerminator* in step 2 (§3.4), and so the string literal is not valid in step 3. Instead, one should write `"\n"` (§3.10.6). Similarly, it is not correct to write `"\u000d"` for a string literal containing a single carriage return (CR). Instead

use `"\r"`.

Each string literal is a reference (§4.3) to an instance (§4.3.1, §12.5) of class `String` (§4.3.3, §20.12). `String` objects have a constant value. String literals-or, more generally, strings that are the values of constant expressions (§15.27)-are "interned" so as to share unique instances, using the method `String.intern` (§20.12.47).

Thus, the test program consisting of the compilation unit (§7.3):

```
package testPackage;
```

```
class Test {
    public static void main(String[] args) {
        String hello = "Hello", lo = "lo";
        System.out.print((hello == "Hello") + " ");
        System.out.print((Other.hello == hello) + " ");
        System.out.print((other.Other.hello == hello) + " ");
        System.out.print((hello == ("Hel"+"lo")) + " ");
        System.out.print((hello == ("Hel"+lo)) + " ");
        System.out.println(hello == ("Hel"+lo).intern());
    }
}

class Other { static String hello = "Hello"; }
```

and the compilation unit:

```
package other;

public class Other { static String hello = "Hello"; }
```

produces the output:

```
true true true true false true
```

This example illustrates six points:

- Literal strings within the same class (§8) in the same package (§7) represent references to the same `String` object (§4.3.1).
- Literal strings within different classes in the same package represent references to the same `String` object.
- Literal strings within different classes in different packages likewise represent references to the same `String` object.

- Strings computed by constant expressions (§15.27) are computed at compile time and then treated as if they were literals.
- Strings computed at run time are newly created and therefore distinct.
- The result of explicitly interning a computed string is the same string as any pre-existing literal string with the same contents.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

3.10.6 Escape Sequences for Character and String Literals

The character and string *escape sequences* allow for the representation of some nongraphic characters as well as the single quote, double quote, and backslash characters in character literals (§3.10.4) and string literals (§3.10.5).

EscapeSequence:

```
\ b          /* \u0008: backspace BS
*/
\ t          /* \u0009: horizontal tab HT
*/
\ n          /* \u000a: linefeed LF
*/
\ f          /* \u000c: form feed FF
*/
\ r          /* \u000d: carriage return CR
*/
\ "          /* \u0022: double quote      "
*/
\ '          /* \u0027: single quote '
*/
\ \          /* \u005c: backslash \
*/
OctalEscape  /* \u0000 to \u00ff: from octal value */
```

OctalEscape:

```
\ OctalDigit
\ OctalDigit OctalDigit
\ ZeroToThree OctalDigit OctalDigit
```

OctalDigit: one of

```
0 1 2 3 4 5 6 7
```

ZeroToThree: one of

```
0 1 2 3
```

It is a compile-time error if the character following a backslash in an escape is not an ASCII b, t, n, f, r, ", ', \, 0, 1, 2, 3, 4, 5, 6, or 7. The Unicode escape \u is processed earlier (§3.3). (Octal escapes are provided for compatibility with C, but can express only Unicode values \u0000 through \u00FF, so Unicode escapes are usually preferred.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


3.10.7 The Null Literal

The null type has one value, the null reference, represented by the literal `null`, which is formed from ASCII characters. A *null literal* is always of the null type.

NullLiteral:

null

{ewl msdncd.dll, ewcright, /c"Microsoft"}

3.11 Separators

The following nine ASCII characters are the Java *separators* (punctuators):

Separator: one of

() { } [] ; , .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

3.12 Operators

The following 37 tokens are the Java *operators*, formed from ASCII characters:

Operator: one of

=	>	<	!	~	?	:				
==	<=	>=	!=	&&		++	--			
+	-	*	/	&		^	%	<<	>>	>>>
+=	-=	*=	/=	&=	=	^=	%=	<<=	>>=	>>>=

Give her no token but stones; for she's as hard as steel.

--William Shakespeare, *Two Gentlemen of Verona*, Act I, scene i

These lords are visited; you are not free;

For the Lord's tokens on you do I see.

--William Shakespeare, *Love's Labour's Lost*, Act V, scene ii

Thou, thou, Lysander, thou hast given her rhymes,

And interchanged love-tokens with my child.

--William Shakespeare, *A Midsummer Night's Dream*, Act I, scene i

Here is a letter from Queen Hecuba,

A token from her daughter . . .

--William Shakespeare, *Troilus and Cressida*, Act V, scene i

Are there no other tokens . . . ?

--William Shakespeare, *Measure for Measure*, Act IV, scene i

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Types, Values, and Variables

*I send no agent or medium,
offer no representative of value,
but offer the value itself.*

--Walt Whitman, Carol of Occupations (1855), in Leaves of Grass

Java is a *strongly typed* language, which means that every variable and every expression has a type that is known at compile time. Types limit the values that a variable (§4.5) can hold or that an expression can produce, limit the operations supported on those values, and determine the meaning of the operations. Strong typing helps detect errors at compile time.

The types of the Java language are divided into two categories: primitive types and reference types. The primitive types (§4.2) are the `boolean` type and the numeric types. The numeric types are the integral types `byte`, `short`, `int`, `long`, and `char`, and the floating-point types `float` and `double`. The reference types (§4.3) are class types, interface types, and array types. There is also a special null type. An object (§4.3.1) in Java is a dynamically created instance of a class type or a dynamically created array. The values of a reference type are references to objects. All objects, including arrays, support the methods of class `Object` (§4.3.2). String literals are represented by `String` objects (§4.3.3).

Types are the same (§4.3.4) if they have the same fully qualified names and are loaded by the same class loader. Names of types are used (§4.4) in declarations, in casts, in class instance creation expressions, in array creation expressions, and in `instanceof` operator expressions.

A variable (§4.5) is a storage location. A variable of a primitive type always holds a value of that exact type. A variable of a class type *T* can hold a null reference or a reference to an instance of class *T* or of any class that is a subclass of *T*. A variable of an interface type can hold a null reference or a reference to any instance of any class that implements the interface. If *T* is a primitive type, then a variable of type "array of *T*" can hold a null reference or a reference to any array of type "array of *T*"; if *T* is a reference type, then a variable of type "array of *T*" can hold a null reference or a reference to any array of type "array of *S*" such that type *S* is assignable (§5.2) to type *T*. A variable of type `Object` can hold a null reference or a reference to any object, whether class instance or array.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.1 The Kinds of Types and Values

There are two kinds of *types* in Java: primitive types (§4.2) and reference types (§4.3). There are, correspondingly, two kinds of data values that can be stored in variables, passed as arguments, returned by methods, and operated on: primitive values (§4.2) and reference values (§4.3).

Type:

PrimitiveType

ReferenceType

There is also a special *null type*, the type of the expression `null`, which has no name. Because the null type has no name, it is impossible to declare a variable of the null type or to cast to the null type. The null reference is the only possible value of an expression of null type. The null reference can always be cast to any reference type. In practice, the Java programmer can ignore the null type and just pretend that `null` is merely a special literal that can be of any reference type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.2 Primitive Types and Values

A *primitive type* is predefined by the Java language and named by its reserved keyword (§3.9):

PrimitiveType:

NumericType

boolean

NumericType:

IntegralType

FloatingPointType

IntegralType: one of

byte short int long char

FloatingPointType: one of

float double

Primitive values do not share state with other primitive values. A variable whose type is a primitive type always holds a primitive value of that same type. The value of a variable of primitive type can be changed only by assignment operations on that variable.

The *numeric types* are the integral types and the floating-point types.

The *integral types* are `byte`, `short`, `int`, and `long`, whose values are 8-bit, 16-bit, 32-bit and 64-bit signed two's-complement integers, respectively, and `char`, whose values are 16-bit unsigned integers representing Unicode characters.

The *floating-point types* are `float`, whose values are 32-bit IEEE 754 floating-point numbers, and `double`, whose values are 64-bit IEEE 754 floating-point numbers.

The `boolean` type has exactly two values: `true` and `false`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

4.2.1 Integral Types and Values

The values of the integral types are integers in the following ranges:

- For `byte`, from -128 to 127, inclusive
- For `short`, from -32768 to 32767, inclusive
- For `int`, from -2147483648 to 2147483647, inclusive
- For `long`, from -9223372036854775808 to 9223372036854775807, inclusive
- For `char`, from `'\u0000'` to `'\uffff'` inclusive, that is, from 0 to 65535

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.2.2 Integer Operations

Java provides a number of operators that act on integral values:

- The comparison operators, which result in a value of type `boolean`:
 - The numerical comparison operators `<`, `<=`, `>`, and `>=` (§15.19.1)
 - The numerical equality operators `==` and `!=` (§15.20.1)
- The numerical operators, which result in a value of type `int` or `long`:
 - The unary plus and minus operators `+` and `-` (§15.14.3, §15.14.4)
 - The multiplicative operators `*`, `/`, and `%` (§15.16)
 - The additive operators `+` and `-` (§15.17.2)
 - The increment operator `++`, both prefix (§15.14.1) and postfix (§15.13.2)
 - The decrement operator `--`, both prefix (§15.14.2) and postfix (§15.13.3)
 - The signed and unsigned shift operators `<<`, `>>`, and `>>>` (§15.18)
 - The bitwise complement operator `~` (§15.14.5)
 - The integer bitwise operators `&`, `|`, and `^` (§15.21.1)
- The conditional operator `? :` (§15.24)
- The cast operator, which can convert from an integral value to a value of any specified numeric type (§5.4, §15.15)
- The string concatenation operator `+` (§15.17.1), which, when given a `String` operand and an integral operand, will convert the integral operand to a `String` representing its value in decimal form, and then produce a newly created `String` that is the concatenation of the two strings

Other useful constructors, methods, and constants are predefined in the classes `Integer` (§20.7), `Long` (§20.8), and `Character` (§20.5).

If an integer operator other than a shift operator has at least one operand of type `long`, then the operation is carried out using 64-bit precision, and the result of the numerical operator is of type `long`. If the other operand is not `long`, it is first widened (§5.1.2) to type `long` by numeric promotion (§5.6). Otherwise, the operation is carried out using 32-bit precision, and the result of the numerical operator is of type `int`. If either operand is not an `int`, it is first widened to type `int` by numeric promotion.

The built-in integer operators do not indicate overflow or underflow in any way. The only numeric operators that can throw an exception (§11) are the integer divide operator `/` (§15.16.2) and the integer remainder operator `%` (§15.16.3), which throw an `ArithmeticException` if the right-hand operand is zero.

The example:

```
class Test {
    public static void main(String[] args) {
        int i = 1000000;
        System.out.println(i * i);
        long l = i;
        System.out.println(l * l);
    }
}
```



```
        System.out.println(20296 / (1 - i));  
    }  
}
```

produces the output:

```
-727379968  
1000000000000
```

and then encounters an `ArithmeticException` in the division by `1 - i`, because `1 - i` is zero. The first multiplication is performed in 32-bit precision, whereas the second multiplication is a `long` multiplication. The value `-727379968` is the decimal value of the low 32 bits of the mathematical result, `10000000000000`, which is a value too large for type `int`.

Any value of any integral type may be cast to or from any numeric type. There are no casts between integral types and the type `boolean`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.2.3 Floating-Point Types and Values

The floating-point types are `float` and `double`, representing the single-precision 32-bit and double-precision 64-bit format IEEE 754 values and operations as specified in *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985 (IEEE, New York).

The IEEE 754 standard includes not only positive and negative sign-magnitude numbers, but also positive and negative zeros, positive and negative *infinities*, and a special *Not-a-Number* (hereafter abbreviated NaN). The NaN value is used to represent the result of certain operations such as dividing zero by zero. NaN constants of both `float` and `double` type are predefined as `Float.NaN` (§20.9.5) and `Double.NaN` (§20.10.5).

The finite nonzero values of type `float` are of the form {`ewc msdn`, `EWGraphic`, `LNG5h 0 /a "langrefANC.BMP"`}, where *s* is +1 or -1, *m* is a positive integer less than {`ewc msdn`, `EWGraphic`, `LNG5h 1 /a "langrefANC1.BMP"`}, and *e* is an integer between -149 and 104, inclusive. Values of that form such that *m* is positive but less than {`ewc msdn`, `EWGraphic`, `LNG5h 2 /a "langrefANC2.BMP"`} and *e* is equal to -149 are said to be *denormalized*.

The finite nonzero values of type `double` are of the form {`ewc msdn`, `EWGraphic`, `LNG5h 3 /a "langrefANC3.BMP"`}, where *s* is +1 or -1, *m* is a positive integer less than {`ewc msdn`, `EWGraphic`, `LNG5h 4 /a "langrefANC4.BMP"`}, and *e* is an integer between -1075 and 970, inclusive. Values of that form such that *m* is positive but less than {`ewc msdn`, `EWGraphic`, `LNG5h 5 /a "langrefANC5.BMP"`} and *e* is equal to -1075 are said to be *denormalized*.

Except for NaN, floating-point values are *ordered*; arranged from smallest to largest, they are negative infinity, negative finite nonzero values, negative zero, positive zero, positive finite nonzero values, and positive infinity.

Positive zero and negative zero compare equal; thus the result of the expression `0.0== -0.0` is `true` and the result of `0.0> -0.0` is `false`. But other operations can distinguish positive and negative zero; for example, `1.0/0.0` has the value positive infinity, while the value of `1.0/-0.0` is negative infinity. The operations `Math.min` and `Math.max` also distinguish positive zero and negative zero.

NaN is *unordered*, so the numerical comparison operators `<`, `<=`, `>`, and `>=` return `false` if either or both operands are NaN (§15.19.1). The equality operator `==` returns `false` if either operand is NaN, and the inequality operator `!=` returns `true` if either operand is NaN (§15.20.1). In particular, `x!=x` is `true` if and only if *x* is NaN, and `(x<y) == !(x>y)` will be `false` if *x* or *y* is NaN.

Any value of a floating-point type may be cast to or from any numeric type. There are no casts between floating-point types and the type `boolean`.

{`ewl msdn`, `ewcright`, `/c"Microsoft"`}

4.2.4 Floating-Point Operations

Java provides a number of operators that act on floating-point values:

- The comparison operators, which result in a value of type `boolean`:
 - The numerical comparison operators `<`, `<=`, `>`, and `>=` (§15.19.1)
 - The numerical equality operators `==` and `!=` (§15.20.1)
- The numerical operators, which result in a value of type `float` or `double`:
 - The unary plus and minus operators `+` and `-` (§15.14.3, §15.14.4)
 - The multiplicative operators `*`, `/`, and `%` (§15.16)
 - The additive operators `+` and `-` (§15.17.2)
 - The increment operator `++`, both prefix (§15.14.1) and postfix (§15.13.2)
 - The decrement operator `--`, both prefix (§15.14.2) and postfix (§15.13.3)
- The conditional operator `? :` (§15.24)
- The cast operator, which can convert from a floating-point value to a value of any specified numeric type (§5.4, §15.15)
- The string concatenation operator `+` (§15.17.1), which, when given a `String` operand and a floating-point operand, will convert the floating-point operand to a `String` representing its value in decimal form (without information loss), and then produce a newly created `String` by concatenating the two strings

Other useful constructors, methods, and constants are predefined in the classes `Float` (§20.9), `Double` (§20.10), and `Math` (§20.11).

If at least one of the operands to a binary operator is of floating-point type, then the operation is a floating-point operation, even if the other is integral.

If at least one of the operands to a numerical operator is of type `double`, then the operation is carried out using 64-bit floating-point arithmetic, and the result of the numerical operator is a value of type `double`. (If the other operand is not a `double`, it is first widened to type `double` by numeric promotion (§5.6).) Otherwise, the operation is carried out using 32-bit floating-point arithmetic, and the result of the numerical operator is a value of type `float`. If the other operand is not a `float`, it is first widened to type `float` by numeric promotion.

Operators on floating-point numbers behave exactly as specified by IEEE 754. In particular, Java requires support of IEEE 754 *denormalized* floating-point numbers and *gradual underflow*, which make it easier to prove desirable properties of particular numerical algorithms. Floating-point operations in Java do not "flush to zero" if the calculated result is a denormalized number.

Java requires that floating-point arithmetic behave as if every floating-point operator rounded its floating-point result to the result precision. *Inexact* results must be rounded to the representable value nearest to the infinitely precise result; if the two nearest representable values are equally near, the one with its least significant bit zero is chosen. This is the IEEE 754 standard's default rounding mode known as *round to nearest*.

Java uses *round toward zero* when converting a floating value to an integer (§5.1.3), which acts, in this case, as though the number were truncated, discarding the mantissa bits. Rounding toward zero chooses at its result the format's value closest to and no greater in magnitude than the infinitely precise result.

Java floating-point operators produce no exceptions (§11). An operation that overflows produces a

signed infinity, an operation that underflows produces a signed zero, and an operation that has no mathematically definite result produces NaN. All numeric operations with NaN as an operand produce NaN as a result. As has already been described, NaN is unordered, so a numeric comparison operation involving one or two NaNs returns `false` and any `!=` comparison involving NaN returns `true`, including `x!=x` when `x` is NaN.

The example program:

```
class Test {

    public static void main(String[] args) {

        // An example of overflow:
        double d = 1e308;
        System.out.print("overflow produces infinity: ");
        System.out.println(d + "*10==" + d*10);

        // An example of gradual underflow:
        d = 1e-305 * Math.PI;
        System.out.print("gradual underflow: " + d + "\n");
        for (int i = 0; i < 4; i++)
            System.out.print(" " + (d /= 100000));
        System.out.println();

        // An example of NaN:
        System.out.print("0.0/0.0 is Not-a-Number: ");
        d = 0.0/0.0;
        System.out.println(d);

        // An example of inexact results and rounding:
        System.out.print("inexact results with float:");
        for (int i = 0; i < 100; i++) {
            float z = 1.0f / i;
            if (z * i != 1.0f)
                System.out.print(" " + i);
        }
        System.out.println();

        // Another example of inexact results and rounding:
        System.out.print("inexact results with double:");
        for (int i = 0; i < 100; i++) {
            double z = 1.0 / i;
            if (z * i != 1.0)
                System.out.print(" " + i);
        }
        System.out.println();

        // An example of cast to integer rounding:
        System.out.print("cast to int rounds toward 0: ");
        d = 12345.6;
        System.out.println((int)d + " " + (int)(-d));
    }
}
```


produces the output:

```
overflow produces infinity: 1.0e+308==Infinity
gradual underflow: 3.141592653589793E-305
    3.1415926535898E-310 3.141592653E-315 3.142E-320 0.0
0.0/0.0 is Not-a-Number: NaN
inexact results with float: 0 41 47 55 61 82 83 94 97
inexact results with double: 0 49 98
cast to int rounds toward 0: 12345 -12345
```

This example demonstrates, among other things, that gradual underflow can result in a gradual loss of precision.

The inexact results when i is 0 involve division by zero, so that z becomes positive infinity, and $z * 0$ is NaN, which is not equal to 1.0.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.2.5 The boolean Type and boolean Values

The `boolean` type represents a logical quantity with two possible values, indicated by the literals `true` and `false` (§3.10.3). The boolean operators are:

- The relational operators `==` and `!=` (§15.20.2)
- The logical-complement operator `!` (§15.14.6)
- The logical operators `&`, `^`, and `|` (§15.21.2)
- The conditional-and and conditional-or operators `&&` (§15.22) and `||` (§15.23)
- The conditional operator `?:` (§15.24)
- The string concatenation operator `+` (§15.17.1), which, when given a `String` operand and a boolean operand, will convert the boolean operand to a `String` (either `"true"` or `"false"`), and then produce a newly created `String` that is the concatenation of the two strings

Boolean expressions determine the control flow in several kinds of statements:

- The `if` statement (§14.8)
- The `while` statement (§14.10)
- The `do` statement (§14.11)
- The `for` statement (§14.12)

A `boolean` expression also determines which subexpression is evaluated in the conditional `?:` operator (§15.24).

Only `boolean` expressions can be used in control flow statements and as the first operand of the conditional operator `?:`. An integer `x` can be converted to a `boolean`, following the C language convention that any nonzero value is `true`, by the expression `x!=0`. An object reference `obj` can be converted to a `boolean`, following the C language convention that any reference other than `null` is `true`, by the expression `obj!=null`.

A cast of a `boolean` value to type `boolean` is allowed (§5.1.1); no other casts on type `boolean` are allowed. A `boolean` can be converted to a string by string conversion (§5.4).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.3 Reference Types and Values

There are three kinds of *reference types*: class types (§8), interface types (§9), and array types (§10).

ReferenceType:

ClassOrInterfaceType

ArrayType

ClassOrInterfaceType:

ClassType

InterfaceType

ClassType:

TypeName

InterfaceType:

TypeName

ArrayType:

Type []

Names are described in §6; type names in §6.5 and, specifically, §6.5.4.

The sample code:

```
class Point { int[] metrics; }
```

```
interface Move { void move(int deltax, int deltay); }
```

declares a class type `Point`, an interface type `Move`, and uses an array type `int[]` (an array of `int`) to declare the field `metrics` of the class `Point`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.3.1 Objects

An *object* is a *class instance* or an array.

The reference values (often just *references*) are *pointers* to these objects, and a special null reference, which refers to no object.

A class instance is explicitly created by a class instance creation expression (§15.8), or by invoking the `newInstance` method of class `Class` (§20.3.8). An array is explicitly created by an array creation expression (§15.8).

A new class instance is implicitly created when the string concatenation operator `+` (§15.17.1) is used in an expression, resulting in a new object of type `String` (§4.3.3, §20.12). A new array object is implicitly created when an array initializer expression (§10.6) is evaluated; this can occur when a class or interface is initialized (§12.4), when a new instance of a class is created (§15.8), or when a local variable declaration statement is executed (§14.3).

Many of these cases are illustrated in the following example:

```
class Point {
    int x, y;
    Point() { System.out.println("default"); }
    Point(int x, int y) { this.x = x; this.y = y; }

    // A Point instance is explicitly created at class initialization
time: static Point origin = new Point(0,0);

    // A String can be implicitly created by a + operator:
    public String toString() {

        return "(" + x + "," + y + ")";

    }
}

class Test {
    public static void main(String[] args) {
        // A Point is explicitly created using newInstance:
        Point p = null;
        try {
            p = (Point)Class.forName("Point").newInstance();
        } catch (Exception e) {
            System.out.println(e);
        }

        // An array is implicitly created by an array constructor:
        Point a[] = { new Point(0,0), new Point(1,1) };

        // Strings are implicitly created by + operators:
        System.out.println("p: " + p);
    }
}
```



```

        System.out.println("a: { " + a[0] + ", "
                                + a[1] +
                                " }");

        // An array is explicitly created by an array creation
expression:
        String sa[] = new String[2];
        sa[0] = "he"; sa[1] = "llo";
        System.out.println(sa[0] + sa[1]);
    }
}

```

which produces the output:

```

default
p: (0,0)
a: { (0,0), (1,1) }
hello

```

The operators on references to objects are:

- Field access, using either a qualified name ([§6.6](#)) or a field access expression ([§15.10](#))
- Method invocation ([§15.11](#))
- The cast operator ([§5.4](#), [§15.15](#))
- The string concatenation operator + ([§15.17.1](#)), which, when given a `String` operand and a reference, will convert the reference to a `String` by invoking the `toString` method ([§20.1.2](#)) of the referenced object (using `"null"` if either the reference or the result of `toString` is a null reference), and then will produce a newly created `String` that is the concatenation of the two strings
- The `instanceof` operator ([§15.19.2](#))
- The reference equality operators `==` and `!=` ([§15.20.3](#))
- The conditional operator `? :` ([§15.24](#)).

There may be many references to the same object. Most objects have state, stored in the fields of objects that are instances of classes or in the variables that are the components of an array object. If two variables contain references to the same object, the state of the object can be modified using one variable's reference to the object, and then the altered state can be observed through the reference in the other variable.

The example program:

```

class Value { int val; }

class Test {

```



```

public static void main(String[] args) {
    int i1 = 3;
    int i2 = i1;
    i2 = 4;
    System.out.print("i1==" + i1);
    System.out.println(" but i2==" + i2);
    Value v1 = new Value();
    v1.val = 5;
    Value v2 = v1;
    v2.val = 6;
    System.out.print("v1.val==" + v1.val);
    System.out.println(" and v2.val==" + v2.val);
}
}

```

produces the output:

```

i1==3 but i2==4
v1.val==6 and v2.val==6

```

because `v1.val` and `v2.val` reference the same instance variable ([§4.5.3](#)) in the one `Value` object created by the only `new` expression, while `i1` and `i2` are different variables.

See [§10](#) and [§15.9](#) for examples of the creation and use of arrays.

Each object has an associated lock ([§17.13](#)), which is used by `synchronized` methods ([§8.4.3](#)) and the `synchronized` statement ([§14.17](#)) to provide control over concurrent access to state by multiple threads ([§17.12](#), [§20.20](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.3.2 The Class Object

The standard class `Object` is a superclass (§8.1) of all other classes. A variable of type `Object` can hold a reference to any object, whether it is an instance of a class or an array (§10). All class and array types inherit the methods of class `Object`, which are summarized here and completely specified in §20.1:

```
package java.lang;

public class Object {
    public final Class getClass() { . . . }
    public String toString() { . . . }
    public boolean equals(Object obj) { . . . }
    public int hashCode() { . . . }
    protected Object clone()
        throws CloneNotSupportedException { . . . }
    public final void wait()

        throws IllegalMonitorStateException,

        InterruptedException { . . . }
    public final void wait(long millis)
        throws IllegalMonitorStateException,
        InterruptedException { . . . }
    public final void wait(long millis, int nanos) { . . . }
        throws IllegalMonitorStateException,
        InterruptedException { . . . }
    public final void notify() { . . . }
        throws IllegalMonitorStateException
    public final void notifyAll() { . . . }
        throws IllegalMonitorStateException
    protected void finalize()
        throws Throwable { . . . }
}
```

The members of `Object` are as follows:

- The method `getClass` returns the `Class` (§20.3) object that represents the class of the object. A `Class` object exists for each reference type. It can be used, for example, to discover the fully qualified name of a class, its members, its immediate superclass, and any interfaces that it implements. A class method that is declared `synchronized` (§8.4.3.5) synchronizes on the lock associated with the `Class` object of the class.
- The method `toString` returns a `String` representation of the object.
- The methods `equals` and `hashCode` are declared for the benefit of hashtables such as `java.util.Hashtable` (§21.7). The method `equals` defines a notion of object equality, which is based on value, not reference, comparison.
- The method `clone` is used to make a duplicate of an object.
- The methods `wait`, `notify`, and `notifyAll` are used in concurrent programming using

threads, as described in §17.

- The method `finalize` is run just before an object is destroyed and is described in §12.6.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.3.3 The Class String

Instances of class `String` ([§20.12](#)) represent sequences of Unicode characters. A `String` object has a constant (unchanging) value. String literals ([§3.10.5](#)) are references to instances of class `String`.

The string concatenation operator `+` ([§15.17.1](#)) implicitly creates a new `String` object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.3.4 When Reference Types Are the Same

Two reference types the *same type* if:

- They are both class or both interface types, are loaded by the same class loader, and have the same fully-qualified name (§6.6), in which case they are sometimes said to be the *same class* or the *same interface*.
- They are both array types, and have the same component type (§10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

4.4 Where Types Are Used

Types are used when they appear in declarations or in certain expressions.

The following code fragment contains one or more instances of each kind of usage of a type:

```
import java.util.Random;

class MiscMath {

    int divisor;

    MiscMath(int divisor) {
        this.divisor = divisor;
    }

    float ratio(long l) {
        try {
            l /= divisor;
        } catch (Exception e) {
            if (e instanceof ArithmeticException)
                l = Long.MAX_VALUE;
            else
                l = 0;
        }
        return (float)l;
    }

    double gausser() {
        Random r = new Random();
        double[] val = new double[2];
        val[0] = r.nextGaussian();
        val[1] = r.nextGaussian();
        return (val[0] + val[1]) / 2;
    }
}
```

In this example, types are used in declarations of the following:

- Imported types ([§7.5](#)); here the type `Random`, imported from the type `java.util.Random` of the package `java.util`, is declared
- Fields, which are the class variables and instance variables of classes ([§8.3](#)), and constants of interfaces ([§9.3](#)); here the field `divisor` in the class `MiscMath` is declared to be of type `int`
- Method parameters ([§8.4.1](#)); here the parameter `l` of the method `ratio` is declared to be of

type long

- Method results (§8.4); here the result of the method `ratio` is declared to be of type `float`, and the result of the method `gausser` is declared to be of type `double`
- Constructor parameters (§8.6.1); here the parameter of the constructor for `MiscMath` is declared to be of type `int`
- Local variables (§14.3, §14.12); the local variables `r` and `val` of the method `gausser` are declared to be of types `Random` and `double[]` (array of `double`)
- Exception handler parameters (§14.18); here the exception handler parameter `e` of the `catch` clause is declared to be of type `Exception`

and in expressions of the following kinds:

- Class instance creations (§15.8); here a local variable `r` of method `gausser` is initialized by a class instance creation expression that uses the type `Random`
- Array creations (§15.9); here the local variable `val` of method `gausser` is initialized by an array creation expression that creates an array of `double` with size 2
- Casts (§15.15); here the `return` statement of the method `ratio` uses the `float` type in a cast
- The `instanceof` operator (§15.19.2); here the `instanceof` operator tests whether `e` is assignment compatible with the type `ArithmeticException`

{ewl msdncd.dll, ewcright, /c"Microsoft"}

4.5 Variables

A variable is a storage location and has an associated type, sometimes called its *compile-time type*, that is either a primitive type (§4.2) or a reference type (§4.3). A variable always contains a value that is assignment compatible (§5.2) with its type. A variable's value is changed by an assignment (§15.25) or by a prefix or postfix ++ (increment) or -- (decrement) operator (§15.13.2, §15.13.3, §15.14.1, §15.14.2).

Compatibility of the value of a variable with its type is guaranteed by the design of the Java language. Default values are compatible (§4.5.4) and all assignments to a variable are checked for assignment compatibility (§5.2), usually at compile time, but, in a single case involving arrays, a run-time check is made (§10.10).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.5.1 Variables of Primitive Type

A variable of a primitive type always holds a value of that exact primitive type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.5.2 Variables of Reference Type

A variable of reference type can hold either of the following:

- A null reference
- A reference to any object (§4.3) whose class (§4.5.5) is assignment compatible (§5.2) with the type of the variable

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.5.3 Kinds of Variables

There are seven kinds of variables:

- A *class variable* is a field declared using the keyword `static` within a class declaration (§8.3.1.1), or with or without the keyword `static` within an interface declaration (§9.3). A class variable is created when its class or interface is loaded (§12.2) and is initialized to a default value (§4.5.4). The class variable effectively ceases to exist when its class or interface is unloaded (§12.8), after any necessary finalization of the class or interface (§12.6) has been completed.
- An *instance variable* is a field declared within a class declaration without using the keyword `static` (§8.3.1.1). If a class `T` has a field `a` that is an instance variable, then a new instance variable `a` is created and initialized to a default value (§4.5.4) as part of each newly created object of class `T` or of any class that is a subclass of `T` (§8.1.3). The instance variable effectively ceases to exist when the object of which it is a field is no longer referenced, after any necessary finalization of the object (§12.6) has been completed.
- *Array components* are unnamed variables that are created and initialized to default values (§4.5.4) whenever a new object that is an array is created (§15.9). The array components effectively cease to exist when the array is no longer referenced. See §10 for a description of arrays.
- *Method parameters* (§8.4.1) name argument values passed to a method. For every parameter declared in a method declaration, a new parameter variable is created each time that method is invoked (§15.11). The new variable is initialized with the corresponding argument value from the method invocation. The method parameter effectively ceases to exist when the execution of the body of the method is complete.
- *Constructor parameters* (§8.6.1) name argument values passed to a constructor. For every parameter declared in a constructor declaration, a new parameter variable is created each time a class instance creation expression (§15.8) or explicit constructor invocation (§8.6.5) invokes that constructor. The new variable is initialized with the corresponding argument value from the creation expression or constructor invocation. The constructor parameter effectively ceases to exist when the execution of the body of the constructor is complete.
- An *exception-handler parameter* is created each time an exception is caught by a `catch` clause of a `try` statement (§14.18). The new variable is initialized with the actual object associated with the exception (§11.3, §14.16). The exception-handler parameter effectively ceases to exist when execution of the block associated with the `catch` clause is complete.
- *Local variables* are declared by local variable declaration statements (§14.3). Whenever the flow of control enters a block (§14.2) or `for` statement (§14.12), a new variable is created for each local variable declared in a local variable declaration statement immediately contained within that block or `for` statement. A local variable declaration statement may contain an expression which initializes the variable. The local variable with an initializing expression is not initialized, however, until the local variable declaration statement that declares it is executed. (The rules of definite assignment (§16) prevent the value of a local variable from being used before it has been initialized or otherwise assigned a value.) The local variable effectively ceases to exist when the execution of the block or `for` statement is complete.

Were it not for one exceptional situation, a local variable could always be regarded as being created when its local variable declaration statement is executed. The exceptional situation involves the `switch` statement (§14.9), where it is possible for control to enter a block but bypass execution of a local variable declaration statement. Because of the restrictions imposed by the rules of definite

assignment (§16), however, the local variable declared by such a bypassed local variable declaration statement cannot be used before it has been definitely assigned a value by an assignment expression (§15.25).

The following example contains several different kinds of variables:

```
class Point {
    static int numPoints;                //
numPoints is a class variable
    int x, y;                          // x and y are
instance variables
    int[] w = new int[10];              //
w[0] is an array component
    int setX(int x) {                  // x is a
method parameter
        int oldx = this.x;             //
oldx is a local variable
        this.x = x;
        return oldx;
    }
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

4.5.4 Initial Values of Variables

Every variable in a Java program must have a value before its value is used:

- Each class variable, instance variable, or array component is initialized with a *default value* when it is created (§15.8, §15.9, §20.3.6):
 - For type `byte`, the default value is zero, that is, the value of `(byte) 0`.
 - For type `short`, the default value is zero, that is, the value of `(short) 0`.
 - For type `int`, the default value is zero, that is, `0`.
 - For type `long`, the default value is zero, that is, `0L`.
 - For type `float`, the default value is positive zero, that is, `0.0f`.
 - For type `double`, the default value is positive zero, that is, `0.0d`.
 - For type `char`, the default value is the null character, that is, `'\u0000'`.
 - For type `boolean`, the default value is `false`.
 - For all reference types (§4.3), the default value is `null`.
- Each method parameter (§8.4.1) is initialized to the corresponding argument value provided by the invoker of the method (§15.11).
- Each constructor parameter (§8.6.1) is initialized to the corresponding argument value provided by a class instance creation expression (§15.8) or explicit constructor invocation (§8.6.5).
- An exception-handler parameter (§14.18) is initialized to the thrown object representing the exception (§11.3, §14.16).
- A local variable (§14.3, §14.12) must be explicitly given a value before it is used, by either initialization (§14.3) or assignment (§15.25), in a way that can be verified by the compiler using the rules for definite assignment (§16).

The example program:

```
class Point {
    static int npoints;
    int x, y;
    Point root;
}

class Test {
    public static void main(String[] args) {
        System.out.println("npoints=" + Point.npoints);
        Point p = new Point();
        System.out.println("p.x=" + p.x + ", p.y=" + p.y);
        System.out.println("p.root=" + p.root);
    }
}
```

prints:


```
npoints=0  
p.x=0, p.y=0  
p.root=null
```

illustrating the default initialization of `npoints`, which occurs when the class `Point` is prepared ([§12.3.2](#)), and the default initialization of `x`, `y`, and `root`, which occurs when a new `Point` is instantiated. See [§12](#) for a full description of all aspects of loading, linking, and initialization of classes and interfaces, plus a description of the instantiation of classes to make new class instances.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


4.5.5 Variables Have Types, Objects Have Classes

Every object belongs to some particular class: the class that was mentioned in the creation expression that produced the object, the class whose class object was used to invoke the `newInstance` method (§20.3.6) to produce the object, or the `String` class for objects implicitly created by the string concatenation operator + (§15.17.1). This class is called the *class of the object*. (Arrays also have a class, as described at the end of this section.) An object is said to be an instance of its class and of all superclasses of its class.

(Sometimes a variable or expression is said to have a "run-time type" but that is an abuse of terminology; it refers to the class of the object referred to by the value of the variable or expression at run time, assuming that the value is not `null`. Properly speaking, type is a compile-time notion. A variable or expression has a type; an object or array has no type, but belongs to a class.)

The type of a variable is always declared, and the type of an expression can be deduced at compile time. The type limits the possible values that the variable can hold or the expression can produce at run time. If a run-time value is a reference that is not `null`, it refers to an object or array that has a class (not a type), and that class will necessarily be compatible with the compile-time type.

Even though a variable or expression may have a compile-time type that is an interface type, there are no instances of interfaces. A variable or expression whose type is an interface type can reference any object whose class implements (§8.1.4) that interface.

Here is an example of creating new objects and of the distinction between the type of a variable and the class of an object:

```
public interface Colorable {
    void setColor(byte r, byte g, byte b);
}

class Point { int x, y; }

class ColoredPoint extends Point implements Colorable {

    byte r, g, b;

    public void setColor(byte rv, byte gv, byte bv) {
        r = rv; g = gv; b = bv;
    }

}

class Test {
    public static void main(String[] args) {
        Point p = new Point();
        ColoredPoint cp = new ColoredPoint();
        p = cp;
        Colorable c = cp;
    }
}
```

In this example:

- The local variable `p` of the method `main` of class `Test` has type `Point` and is initially assigned a reference to a new instance of class `Point`.
- The local variable `cp` similarly has as its type `ColoredPoint`, and is initially assigned a reference to a new instance of class `ColoredPoint`.
- The assignment of the value of `cp` to the variable `p` causes `p` to hold a reference to a `ColoredPoint` object. This is permitted because `ColoredPoint` is a subclass of `Point`, so the class `ColoredPoint` is assignment compatible (§5.2) with the type `Point`. A `ColoredPoint` object includes support for all the methods of a `Point`. In addition to its particular fields `r`, `g`, and `b`, it has the fields of class `Point`, namely `x` and `y`.
- The local variable `c` has as its type the interface type `Colorable`, so it can hold a reference to any object whose class implements `Colorable`; specifically, it can hold a reference to a `ColoredPoint`.
- Note that an expression such as `"new Colorable()"` is not valid because it is not possible to create an instance of an interface, only of a class.

Every array also has a class; the method `getClass` (§20.1.1), when invoked for an array object, will return a class object (of class `Class`) that represents the class of the array. The classes for arrays have strange names that are not valid Java identifiers; for example, the class for an array of `int` components has the name `"[I"` and so the value of the expression:

```
new int[10].getClass().getName()
```

is the string `"[I"`; see §20.1.1 for details.

*Of on the dappled turf at ease
I sit, and play with similes,
Loose types of things through all degrees.
--William Wordsworth, To the Same Flower*

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Conversions and Promotions

*Thou art not for the fashion of these times,
Where none will sweat but for promotion.*

--William Shakespeare, As You Like It, Act II, scene iii

Every Java expression has a type that can be deduced from the structure of the expression and the types of the literals, variables, and methods mentioned in the expression. It is possible, however, to write an expression in a context where the type of the expression is not appropriate. In some cases, this leads to an error at compile time; for example, if the expression in an `if` statement (§14.8) has any type other than `boolean`, a compile-time error occurs. In other cases, the context may be able to accept a type that is related to the type of the expression; as a convenience, rather than requiring the programmer to indicate a type conversion explicitly, the Java language performs an implicit *conversion* from the type of the expression to a type acceptable for its surrounding context.

A specific conversion from type *S* to type *T* allows an expression of type *S* to be treated at compile time as if it had type *T* instead. In some cases this will require a corresponding action at run time to check the validity of the conversion or to translate the run-time value of the expression into a form appropriate for the new type *T*. For example:

- A conversion from type `Object` (§20.1) to type `Thread` (§20.20) requires a run-time check to make sure that the run-time value is actually an instance of class `Thread` or one of its subclasses; if it is not, an exception is thrown.
- A conversion from type `Thread` to type `Object` requires no run-time action; `Thread` is a subclass of `Object`, so any reference produced by an expression of type `Thread` is a valid reference value of type `Object`.
- A conversion from type `int` to type `long` requires run-time sign-extension of a 32-bit integer value to the 64-bit `long` representation. No information is lost.
- A conversion from type `double` to type `long` requires a nontrivial translation from a 64-bit floating-point value to the 64-bit integer representation. Depending on the actual run-time value, information may be lost.

In every conversion context, only certain specific conversions are permitted. The specific conversions that are possible in Java are grouped for convenience of description into several broad categories:

- Identity conversions
- Widening primitive conversions
- Narrowing primitive conversions
- Widening reference conversions
- Narrowing reference conversions
- String conversions

There are five *conversion contexts* in which conversion of Java expressions may occur. Each context allows conversions in some of the categories named above but not others. The term "conversion" is also used to describe the process of choosing a specific conversion for such a context. For example, we say that an expression that is an actual argument in a method invocation is subject to "method invocation conversion," meaning that a specific conversion will be implicitly chosen for that expression according to the rules for the method invocation argument context.

One conversion context is the operand of a numeric operator such as `+` or `*`. The conversion process for such operands is called *numeric promotion*. Promotion is special in that, in the case of binary

operators, the conversion chosen for one operand may depend in part on the type of the other operand expression.

This chapter first describes the six categories of conversions ([§5.1](#)), including the special conversions to `String` allowed for the string concatenation operator `+`. Then the five conversion contexts are described:

- Assignment conversion ([§5.2](#), [§15.25](#)) converts the type of an expression to the type of a specified variable. The conversions permitted for assignment are limited in such a way that assignment conversion never causes an exception.
- Method invocation conversion ([§5.3](#), [§15.8](#), [§15.11](#)) is applied to each argument in a method or constructor invocation and, except in one case, performs the same conversions that assignment conversion does. Method invocation conversion never causes an exception.
- Casting conversion ([§5.4](#)) converts the type of an expression to a type explicitly specified by a cast operator ([§15.15](#)). It is more inclusive than assignment or method invocation conversion, allowing any specific conversion other than a string conversion, but certain casts to a reference type may cause an exception at run time.
- String conversion ([§5.4](#), [§15.17.1](#)) allows any type to be converted to type `String`.
- Numeric promotion ([§5.6](#)) brings the operands of a numeric operator to a common type so that an operation can be performed.

Here are some examples of the various contexts for conversion:

```
class Test {  
  
    public static void main(String[] args) {  
  
        // Casting conversion (§5.4) of a float literal to  
        // type int. Without the cast operator, this would  
        // be a compile-time error, because this is a  
        // narrowing conversion (§5.1.3):  
        int i = (int)12.5f;  
  
        // String conversion (§5.4) of i's int value:  
        System.out.println("(int)12.5f==" + i);  
  
        // Assignment conversion (§5.2) of i's value to type  
        // float. This is a widening conversion (§5.1.2):  
        float f = i;  
  
        // String conversion of f's float value:  
        System.out.println("after float widening: " + f);  
  
        // Numeric promotion (§5.6) of i's value to type  
        // float. This is a binary numeric promotion.  
        // After promotion, the operation is float*float:  
        System.out.print(f);  
    }  
}
```



```

f = f * i;

// Two string conversions of i and f:
System.out.println("i + f == " + f);

// Method invocation conversion (§5.3) of f's value
// to type double, needed because the method Math.sin
// accepts only a double argument:
double d = Math.sin(f);

// Two string conversions of f and d:
System.out.println("Math.sin(" + f + ") == " + d);
}
}

```

which produces the output:

```

(int)12.5f==12
after float widening: 12.0
12.0*12==144.0
Math.sin(144.0)==-0.49102159389846934

```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1 Kinds of Conversion

Specific type conversions in Java are divided into six categories.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1.1 Identity Conversions

A conversion from a type to that same type is permitted for any type. This may seem trivial, but it has two practical consequences. First, it is always permitted for an expression to have the desired type to begin with, thus allowing the simply stated rule that every expression is subject to conversion, if only a trivial identity conversion. Second, it implies that it is permitted for a program to include redundant cast operators for the sake of clarity.

The only permitted conversion that involves the type `boolean` is the identity conversion from `boolean` to `boolean`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1.2 Widening Primitive Conversions

The following 19 specific conversions on primitive types are called the *widening primitive conversions*:

- `byte` to `short`, `int`, `long`, `float`, or `double`
- `short` to `int`, `long`, `float`, or `double`
- `char` to `int`, `long`, `float`, or `double`
- `int` to `long`, `float`, or `double`
- `long` to `float` or `double`
- `float` to `double`

Widening primitive conversions do not lose information about the overall magnitude of a numeric value. Indeed, conversions widening from an integral type to another integral type and from `float` to `double` do not lose any information at all; the numeric value is preserved exactly. Conversion of an `int` or a `long` value to `float`, or of a `long` value to `double`, may result in *loss of precision*—that is, the result may lose some of the least significant bits of the value. In this case, the resulting floating-point value will be a correctly rounded version of the integer value, using IEEE 754 round-to-nearest mode ([§4.2.4](#)).

A widening conversion of a signed integer value to an integral type *T* simply sign-extends the two's-complement representation of the integer value to fill the wider format. A widening conversion of a character to an integral type *T* zero-extends the representation of the character value to fill the wider format.

Despite the fact that loss of precision may occur, widening conversions among primitive types never result in a run-time exception ([§11](#)).

Here is an example of a widening conversion that loses precision:

```
class Test {
    public static void main(String[] args) {
        int big = 1234567890;
        float approx = big;
        System.out.println(big - (int)approx);
    }
}
```

which prints:

-46

thus indicating that information was lost during the conversion from type `int` to type `float` because values of type `float` are not precise to nine significant digits.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

5.1.3 Narrowing Primitive Conversions

The following 23 specific conversions on primitive types are called the *narrowing primitive conversions*:

- `byte` to `char`
- `short` to `byte` or `char`
- `char` to `byte` or `short`
- `int` to `byte`, `short`, or `char`
- `long` to `byte`, `short`, `char`, or `int`
- `float` to `byte`, `short`, `char`, `int`, or `long`
- `double` to `byte`, `short`, `char`, `int`, `long`, or `float`

Narrowing conversions may lose information about the overall magnitude of a numeric value and may also lose precision.

A narrowing conversion of a signed integer to an integral type *T* simply discards all but the *n* lowest order bits, where *n* is the number of bits used to represent type *T*. In addition to a possible loss of information about the magnitude of the numeric value, this may cause the sign of the resulting value to differ from the sign of the input value.

A narrowing conversion of a character to an integral type *T* likewise simply discards all but the *n* lowest order bits, where *n* is the number of bits used to represent type *T*. In addition to a possible loss of information about the magnitude of the numeric value, this may cause the resulting value to be a negative number, even though characters represent 16-bit unsigned integer values.

A narrowing conversion of a floating-point number to an integral type *T* takes two steps:

- In the first step, the floating-point number is converted either to a `long`, if *T* is `long`, or to an `int`, if *T* is `byte`, `short`, `char`, or `int`, as follows:
 - If the floating-point number is NaN (§4.2.3), the result of the first step of the conversion is an `int` or `long` 0.
 - Otherwise, if the floating-point number is not an infinity, the floating-point value is rounded to an integer value *V*, rounding toward zero using IEEE 754 round-toward-zero mode (§4.2.3). Then there are two cases:
 - If *T* is `long`, and this integer value can be represented as a `long`, then the result of the first step is the `long` value *V*.
 - Otherwise, if this integer value can be represented as an `int`, then the result of the first step is the `int` value *V*.
 - Otherwise, one of the following two cases must be true:
 - The value must be too small (a negative value of large magnitude or negative infinity), and the result of the first step is the smallest representable value of type `int` or `long`.
 - The value must be too large (a positive value of large magnitude or positive infinity), and the result of the first step is the largest representable value of type `int` or `long`.
- In the second step:
 - If *T* is `int` or `long`, the result of the conversion is the result of the first step.
 - If *T* is `byte`, `char`, or `short`, the result of the conversion is the result of a narrowing conversion to type *T* (§5.1.3) of the result of the first step.

The example:

```
class Test {
    public static void main(String[] args) {
        float fmin = Float.NEGATIVE_INFINITY;
        float fmax = Float.POSITIVE_INFINITY;
        System.out.println("long: " + (long)fmin +
                           "..." + (long)fmax);
        System.out.println("int: " + (int)fmin +
                           "..." + (int)fmax);
        System.out.println("short: " + (short)fmin +
                           "..." + (short)fmax);
        System.out.println("char: " + (int)(char)fmin +
                           "..." + (int)(char)fmax);
        System.out.println("byte: " + (byte)fmin +
                           "..." + (byte)fmax);
    }
}
```

produces the output:

```
long: -9223372036854775808..9223372036854775807
int: -2147483648..2147483647
short: 0..-1
char: 0..65535
byte: 0..-1
```

The results for `char`, `int`, and `long` are unsurprising, producing the minimum and maximum representable values of the type.

The results for `byte` and `short` lose information about the sign and magnitude of the numeric values and also lose precision. The results can be understood by examining the low order bits of the minimum and maximum `int`. The minimum `int` is, in hexadecimal, `0x80000000`, and the maximum `int` is `0x7fffffff`. This explains the `short` results, which are the low 16 bits of these values, namely, `0x0000` and `0xffff`; it explains the `char` results, which also are the low 16 bits of these values, namely, `'\u0000'` and `'\uffff'`; and it explains the `byte` results, which are the low 8 bits of these values, namely, `0x00` and `0xff`.

A narrowing conversion from `double` to `float` behaves in accordance with IEEE 754. The result is correctly rounded using IEEE 754 round-to-nearest mode. A value too small to be represented as a `float` is converted to positive or negative zero; a value too large to be represented as a `float` is converted to a (positive or negative) infinity. A `double` NaN is always converted to a `float` NaN.

Despite the fact that overflow, underflow, or other loss of information may occur, narrowing conversions among primitive types never result in a run-time exception (§11).

Here is a small test program that demonstrates a number of narrowing conversions that lose information:

```
class Test {
```



```

public static void main(String[] args) {

    // A narrowing of int to short loses high bits:
    System.out.println("(short)0x12345678==0x" +
        Integer.toHexString((short)0x12345678));

    // A int value not fitting in byte changes sign and magnitude:
    System.out.println("(byte)255==" + (byte)255);

    // A float value too big to fit gives largest int value:
    System.out.println("(int)1e20f==" + (int)1e20f);

    // A NaN converted to int yields zero:
    System.out.println("(int)NaN==" + (int)Float.NaN);

    // A double value too large for float yields infinity:
    System.out.println("(float)-1e100==" + (float)-1e100);

    // A double value too small for float underflows to zero:
    System.out.println("(float)1e-50==" + (float)1e-50);

}
}

```

This test program produces the following output:

```

(short)0x12345678==0x5678
(byte)255==-1
(int)1e20f==2147483647
(int)NaN==0
(float)-1e100==--Infinity
(float)1e-50==0.0

```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

5.1.4 Widening Reference Conversions

The following conversions are called the *widening reference conversions*:

- From any class type *S* to any class type *T*, provided that *S* is a subclass of *T*. (An important special case is that there is a widening conversion to the class type `Object` from any other class type.)
- From any class type *S* to any interface type *K*, provided that *S* implements *K*.
- From the null type to any class type, interface type, or array type.
- From any interface type *J* to any interface type *K*, provided that *J* is a subinterface of *K*.
- From any interface type to type `Object`.
- From any array type to type `Object`.
- From any array type to type `Cloneable`.
- From any array type *SC*[] to any array type *TC*[], provided that *SC* and *TC* are reference types and there is a widening conversion from *SC* to *TC*.

Such conversions never require a special action at run time and therefore never throw an exception at run time. They consist simply in regarding a reference as having some other type in a manner that can be proved correct at compile time.

See [§8](#) for the detailed specifications for classes, [§9](#) for interfaces, and [§10](#) for arrays.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1.5 Narrowing Reference Conversions

The following conversions are called the *narrowing reference conversions*:

- From any class type *S* to any class type *T*, provided that *S* is a superclass of *T*. (An important special case is that there is a narrowing conversion from the class type `Object` to any other class type.)
- From any class type *S* to any interface type *K*, provided that *S* is not `final` and does not implement *K*. (An important special case is that there is a narrowing conversion from the class type `Object` to any interface type.)
- From type `Object` to any array type.
- From type `Object` to any interface type.
- From any interface type *J* to any class type *T* that is not `final`.
- From any interface type *J* to any class type *T* that is `final`, provided that *T* implements *J*.
- From any interface type *J* to any interface type *K*, provided that *J* is not a subinterface of *K* and there is no method name *m* such that *J* and *K* both declare a method named *m* with the same signature but different return types.
- From any array type *SC*[] to any array type *TC*[], provided that *SC* and *TC* are reference types and there is a narrowing conversion from *SC* to *TC*.

Such conversions require a test at run time to find out whether the actual reference value is a legitimate value of the new type. If not, then a `ClassCastException` is thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1.6 String Conversions

There is a string conversion to type `String` from every other type, including the null type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.1.7 Forbidden Conversions

- There is no permitted conversion from any reference type to any primitive type.
- Except for the string conversions, there is no permitted conversion from any primitive type to any reference type.
- There is no permitted conversion from the null type to any primitive type.
- There is no permitted conversion to the null type other than the identity conversion.
- There is no permitted conversion to the type `boolean` other than the identity conversion.
- There is no permitted conversion from the type `boolean` other than the identity conversion and string conversion.
- There is no permitted conversion other than string conversion from class type *S* to a different class type *T* if *S* is not a subclass of *T* and *T* is not a subclass of *S*.
- There is no permitted conversion from class type *S* to interface type *K* if *S* is `final` and does not implement *K*.
- There is no permitted conversion from class type *S* to any array type if *S* is not `Object`.
- There is no permitted conversion other than string conversion from interface type *J* to class type *T* if *T* is `final` and does not implement *J*.
- There is no permitted conversion from interface type *J* to interface type *K* if *J* and *K* declare methods with the same signature but different return types.
- There is no permitted conversion from any array type to any class type other than `Object` or `String`.
- There is no permitted conversion from any array type to any interface type, except to the interface type `Cloneable`, which is implemented by all arrays.
- There is no permitted conversion from array type *SC*[] to array type *TC*[] if there is no permitted conversion other than a string conversion from *SC* to *TC*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

5.2 Assignment Conversion

Assignment conversion occurs when the value of an expression is assigned (§15.25) to a variable: the type of the expression must be converted to the type of the variable. Assignment contexts allow the use of an identity conversion (§5.1.1), a widening primitive conversion (§5.1.2), or a widening reference conversion (§5.1.4). In addition, a narrowing primitive conversion may be used if all of the following conditions are satisfied:

- The expression is a constant expression of type `int`.
- The type of the variable is `byte`, `short`, or `char`.
- The value of the expression (which is known at compile time, because it is a constant expression) is representable in the type of the variable.

If the type of the expression cannot be converted to the type of the variable by a conversion permitted in an assignment context, then a compile-time error occurs.

If the type of an expression can be converted to the type a variable by assignment conversion, we say the expression (or its value) is *assignable* to the variable or, equivalently, that the type of the expression is *assignment compatible* with the type of the variable.

An assignment conversion never causes an exception. (Note, however, that an assignment may result in an exception in a special case involving array elements -see §10.10 and §15.25.1.)

The compile-time narrowing of constants means that code such as:

```
byte theAnswer = 42;
```

is allowed. Without the narrowing, the fact that the integer literal 42 has type `int` would mean that a cast to `byte` would be required:

```
byte theAnswer = (byte)42;
    // cast is permitted but not required
```

A value of primitive type must not be assigned to a variable of reference type; an attempt to do so will result in a compile-time error. A value of type `boolean` can be assigned only to a variable of type `boolean`.

The following test program contains examples of assignment conversion of primitive values:

```
class Test {
    public static void main(String[] args) {
        short s = 12; // narrow
12 to short
        float f = s; // widen
short to float
        System.out.println("f=" + f);

        char c = '\u0123';
        long l = c; // widen char
to long
    }
```



```

        System.out.println("l=0x" + Long.toString(l,16));

        f = 1.23f;
        double d = f;                                // widen
float to double
        System.out.println("d=" + d);
    }
}

```

It produces the following output:

```

f=12.0
i=0x123
d=1.2300000190734863

```

The following test, however, produces compile-time errors:

```

class Test {
    public static void main(String[] args) {
        short s = 123;
        char c = s;                                // error: would
require cast
        s = c;                                    // error: would
require cast
    }
}

```

because not all short values are char values, and neither are all char values short values.

A value of reference type must not be assigned to a variable of primitive type; an attempt to do so will result in a compile-time error.

A value of the null type (the null reference is the only such value) may be assigned to any reference type, resulting in a null reference of that type.

Here is a sample program illustrating assignments of references:

```

public class Point { int x, y; }

public class Point3D extends Point { int z; }

public interface Colorable {
    void setColor(int color);
}

public class ColoredPoint extends Point implements Colorable
{
    int color;
}

```



```

    public void setColor(int color) { this.color = color; }
}

```

```

class Test {

    public static void main(String[] args) {

        // Assignments to variables of class type:
        Point p = new Point();
        p = new Point3D(); //
ok: because Point3d is a // subclass of Point

        Point3D p3d = p; // error:
will require a cast because a // Point might not be
a Point3D // (even though it
is, dynamically, // in this example.)

        // Assignments to variables of type Object:
        Object o = p; // ok:
any object to Object
        int[] a = new int[3];
        Object o2 = a; // ok: an
array to Object

        // Assignments to variables of interface type:
        ColoredPoint cp = new ColoredPoint();
        Colorable c = cp; // ok:
ColoredPoint implements // Colorable

        // Assignments to variables of array type:
        byte[] b = new byte[4];
        a = b; // error: these
are not arrays // of the same
primitive type
        Point3D[] p3da = new Point3D[3];
        Point[] pa = p3da; //
ok: since we can assign a // Point3D to a Point
        p3da = pa; // error: (cast
needed) since a Point // can't be assigned
to a Point3D

```



```

    }
}

```

Assignment of a value of compile-time reference type *S* (source) to a variable of compile-time reference type *T* (target) is checked as follows:

- If *S* is a class type:
 - If *T* is a class type, then *S* must either be the same class as *T*, or *S* must be a subclass of *T*, or a compile-time error occurs.
 - If *T* is an interface type, then *S* must implement interface *T*, or a compile-time error occurs.
 - If *T* is an array type, then a compile-time error occurs.
- If *S* is an interface type:
 - If *T* is a class type, then *T* must be `Object`, or a compile-time error occurs.
 - If *T* is an interface type, then *T* must be either the same interface as *S* or a superinterface of *S*, or a compile-time error occurs.
 - If *T* is an array type, then a compile-time error occurs.
- If *S* is an array type *SC*[], that is, an array of components of type *SC*:
 - If *T* is a class type, then *T* must be `Object`, or a compile-time error occurs.
 - If *T* is an interface type, then a compile-time error occurs unless *T* is the interface type `Cloneable`, the only interface implemented by arrays.
 - If *T* is an array type *TC*[], that is, an array of components of type *TC*, then a compile-time error occurs unless one of the following is true:
 - *TC* and *SC* are the same primitive type.
 - *TC* and *SC* are both reference types and type *SC* is assignable to *TC*, as determined by a recursive application of these compile-time rules for assignability.

See [§8](#) for the detailed specifications for classes, [§9](#) for interfaces, and [§10](#) for arrays.

The following test program illustrates assignment conversions on reference values, but fails to compile because it violates the preceding rules, as described in its comments. This example should be compared to the preceding one.

```

public class Point { int x, y; }

public interface Colorable { void setColor(int color); }

public class ColoredPoint extends Point implements Colorable
{
    int color;
    public void setColor(int color) { this.color = color; }
}

class Test {

    public static void main(String[] args) {

```



```

    Point p = new Point();

    ColoredPoint cp = new ColoredPoint();

    // Okay because ColoredPoint is a subclass of Point:
    p = cp;

    // Okay because ColoredPoint implements Colorable:
    Colorable c = cp;

    // The following cause compile-time errors because
    // we cannot be sure they will succeed, depending on
    // the run-time type of p; a run-time check will be
    // necessary for the needed narrowing conversion and
    // must be indicated by including a cast:
    cp = p; // p might be neither a
ColoredPoint // nor a subclass of ColoredPoint
    c = p; // p might not implement Colorable
}
}

```

Here is another example involving assignment of array objects:

```

class Point { int x, y; }

class ColoredPoint extends Point { int color; }

class Test {
    public static void main(String[] args) {
        long[] veclong = new long[100];
        Object o = veclong;
        // okay
        Long l = veclong;
        // compile-time error
        short[] vecshort = veclong;
        // compile-time error
        Point[] pvec = new Point[100];
        ColoredPoint[] cpvec = new ColoredPoint[100];
        pvec = cpvec;
        // okay
        pvec[0] = new Point();
        // okay at compile time,
    }
}

```

would throw an // but

exception at run time //

cpvec = pvec;


```

        // compile-time error
    }
}

```

In this example:

- The value of `veclong` cannot be assigned to a `Long` variable, because `Long` is a class type (§20.8) other than `Object`. An array can be assigned only to a variable of a compatible array type, or to a variable of type `Object`.
- The value of `veclong` cannot be assigned to `vecshort`, because they are arrays of primitive type, and `short` and `long` are not the same primitive type.
- The value of `cpvec` can be assigned to `pvec`, because any reference that could be the value of an expression of type `ColoredPoint` can be the value of a variable of type `Point`. The subsequent assignment of the new `Point` to a component of `pvec` then would throw an `ArrayStoreException` (if the program were otherwise corrected so that it could be compiled), because a `ColoredPoint` array can't have an instance of `Point` as the value of a component.
- The value of `pvec` cannot be assigned to `cpvec`, because not every reference that could be the value of an expression of type `ColoredPoint` can correctly be the value of a variable of type `Point`. If the value of `pvec` at run time were a reference to an instance of `Point[]`, and the assignment to `cpvec` were allowed, a simple reference to a component of `cpvec`, say, `cpvec[0]`, could return a `Point`, and a `Point` is not a `ColoredPoint`. Thus to allow such an assignment would allow a violation of the type system. A cast may be used (§5.4, §15.15) to ensure that `pvec` references a `ColoredPoint[]`:

```

        cpvec = (ColoredPoint[])pvec;
                // okay, but may throw an
exception at run time

```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.3 Method Invocation Conversion

Method invocation conversion is applied to each argument value in a method or constructor invocation (§15.8, §15.11): the type of the argument expression must be converted to the type of the corresponding parameter. Method invocation contexts allow the use of an identity conversion (§5.1.1), a widening primitive conversion (§5.1.2), or a widening reference conversion (§5.1.4).

Method invocation conversions specifically do not include the implicit narrowing of integer constants which is part of assignment conversion (§5.2). The Java designers felt that including these implicit narrowing conversions would add additional complexity to the overloaded method matching resolution process (§15.11.2). Thus, the example:

```
class Test {  
  
    static int m(byte a, int b) { return a+b; }  
  
    static int m(short a, short b) { return a-b; }  
  
    public static void main(String[] args) {  
        System.out.println(m(12, 2));  
        // compile-time error  
    }  
}
```

causes a compile-time error because the integer literals 12 and 2 have type `int`, so neither method `m` matches under the rules of (§15.11.2). A language that included implicit narrowing of integer constants would need additional rules to resolve cases like this example.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.4 String Conversion

String conversion applies only to the operands of the binary + operator when one of the arguments is a `String`. In this single special case, the other argument to the + is converted to a `String`, and a new `String` which is the concatenation of the two strings is the result of the +. String conversion is specified in detail within the description of the string concatenation + operator ([§15.17.1](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.5 Casting Conversion

Sing away sorrow, cast away care.

-Miguel de Cervantes (1547-1616),
Don Quixote (Lockhart's translation), Chapter viii

Casting conversion is applied to the operand of a cast operator (§15.15): the type of the operand expression must be converted to the type explicitly named by the cast operator. Casting contexts allow the use of an identity conversion (§5.1.1), a widening primitive conversion (§5.1.2), a narrowing primitive conversion (§5.1.3), a widening reference conversion (§5.1.4), or a narrowing reference conversion (§5.1.5). Thus casting conversions are more inclusive than assignment or method invocation conversions: a cast can do any permitted conversion other than a string conversion.

Some casts can be proven incorrect at compile time; such casts result in a compile-time error.

A value of a primitive type can be cast to another primitive type by identity conversion, if the types are the same, or by a widening primitive conversion or a narrowing primitive conversion.

A value of a primitive type cannot be cast to a reference type by casting conversion, nor can a value of a reference type be cast to a primitive type.

The remaining cases involve conversion between reference types. The detailed rules for compile-time correctness checking of a casting conversion of a value of compile-time reference type *S* (source) to a compile-time reference type *T* (target) are as follows:

- If *S* is a class type:
 - If *T* is a class type, then *S* and *T* must be related classes—that is, *S* and *T* must be the same class, or *S* a subclass of *T*, or *T* a subclass of *S*; otherwise a compile-time error occurs.
 - If *T* is an interface type:
 - If *S* is not a `final` class (§8.1.2), then the cast is always correct at compile time (because even if *S* does not implement *T*, a subclass of *S* might).
 - If *S* is a `final` class (§8.1.2), then *S* must implement *T*, or a compile-time error occurs.
 - If *T* is an array type, then *S* must be the class `Object`, or a compile-time error occurs.
- If *S* is an interface type:
 - If *T* is a class type that is not `final` (§8.1.2), then the cast is always correct at compile time (because even if *T* does not implement *S*, a subclass of *T* might).
 - If *T* is a class type that is `final` (§8.1.2), then *T* must implement *S*, or a compile-time error occurs.
 - If *T* is an interface type and if *T* and *S* contain methods with the same signature (§8.4.2) but different return types, then a compile-time error occurs.
- If *S* is an array type `SC[]`, that is, an array of components of type *SC*:
 - If *T* is a class type, then if *T* is not `Object`, then a compile-time error occurs (because `Object` is the only class type to which arrays can be assigned).
 - If *T* is an interface type, then a compile-time error occurs unless *T* is the interface type `Cloneable`, the only interface implemented by arrays.
 - If *T* is an array type `TC[]`, that is, an array of components of type *TC*, then a compile-time error occurs unless one of the following is true:

- *TC* and *SC* are the same primitive type.
- *TC* and *SC* are reference types and type *SC* can be cast to *TC* by a recursive application of these compile-time rules for casting.

See §8 for the detailed specifications of classes, §9 for interfaces, and §10 for arrays.

If a cast to a reference type is not a compile-time error, there are two cases:

- The cast can be determined to be correct at compile time. A cast from the compile-time type *S* to compile-time type *T* is correct at compile time if and only if *S* can be converted to *T* by assignment conversion (§5.2).
 - The cast requires a run-time validity check. If the value at run time is `null`, then the cast is allowed. Otherwise, let *R* be the class of the object referred to by the run-time reference value, and let *T* be the type named in the cast operator. A cast conversion must check, at run time, that the class *R* is assignment compatible with the type *T*, using the algorithm specified in §5.2 but using the class *R* instead of the compile-time type *S* as specified there. (Note that *R* cannot be an interface when these rules are first applied for any given cast, but *R* may be an interface if the rules are applied recursively because the run-time reference value refers to an array whose element type is an interface type.) This modified algorithm is shown here:
 - If *R* is an ordinary class (not an array class):
 - If *T* is a class type, then *R* must be either the same class (§4.3.4) as *T* or a subclass of *T*, or a run-time exception is thrown.
 - If *T* is an interface type, then *R* must implement (§8.1.4) interface *T*, or a run-time exception is thrown.
 - If *T* is an array type, then a run-time exception is thrown.
 - If *R* is an interface:
 - If *T* is a class type, then *T* must be `Object` (§4.3.2, §20.1), or a run-time exception is thrown.
 - If *T* is an interface type, then *R* must be either the same interface as *T* or a subinterface of *T*, or a run-time exception is thrown.
 - If *T* is an array type, then a run-time exception is thrown.
 - If *R* is a class representing an array type *RC*[] -that is, an array of components of type *RC*:
 - If *T* is a class type, then *T* must be `Object` (§4.3.2, §20.1), or a run-time exception is thrown.
 - If *T* is an interface type, then a run-time exception is thrown unless *T* is the interface type `Cloneable`, the only interface implemented by arrays (this case could slip past the compile-time checking if, for example, a reference to an array were stored in a variable of type `Object`).
 - If *T* is an array type *TC*[], that is, an array of components of type *TC*, then a run-time exception is thrown unless one of the following is true:
 - ***TC* and *RC* are the same primitive type.**
 - ***TC* and *RC* are reference types and type *RC* can be cast to *TC* by a recursive application of these run-time rules for casting.**
-

If a run-time exception is thrown, it is a `ClassCastException` ([§11.5.1.1](#), [§20.22](#)).

Here are some examples of casting conversions of reference types, similar to the example in [§5.2](#):

```
public class Point { int x, y; }

public interface Colorable { void setColor(int color); }

public class ColoredPoint extends Point implements Colorable
{
    int color;
    public void setColor(int color) { this.color = color; }
}

final class EndPoint extends Point { }

class Test {
    public static void main(String[] args) {
        Point p = new Point();
        ColoredPoint cp = new ColoredPoint();
        Colorable c;

        // The following may cause errors at run time because
        // we cannot be sure they will succeed; this possibility
        // is suggested by the casts:
        cp = (ColoredPoint)p; //
p might not reference an // object which is a
ColoredPoint // or a subclass of
ColoredPoint
        c = (Colorable)p; // p
might not be Colorable

        // The following are incorrect at compile time because
        // they can never succeed as explained in the text:
        Long l = (Long)p; //
compile-time error #1
        EndPoint e = new EndPoint();
        c = (Colorable)e; //
compile-time error #2
    }
}
```

Here the first compile-time error occurs because the class types `Long` and `Point` are unrelated (that is, they are not the same, and neither is a subclass of the other), so a cast between them will always

fail.

The second compile-time error occurs because a variable of type `EndPoint` can never reference a value that implements the interface `Colorable`. This is because `EndPoint` is a `final` type, and a variable of a `final` type always holds a value of the same run-time type as its compile-time type. Therefore, the run-time type of variable `e` must be exactly the type `EndPoint`, and type `EndPoint` does not implement `Colorable`.

Here is an example involving arrays ([§10](#)):

```
class Point {
    int x, y;

    Point(int x, int y) { this.x = x; this.y = y; }

    public String toString() { return "("+x+","+y+")"; }
}

public interface Colorable { void setColor(int color); }

public class ColoredPoint extends Point implements Colorable
{
    int color;

    ColoredPoint(int x, int y, int color) {
        super(x, y); setColor(color);
    }

    public void setColor(int color) { this.color = color; }

    public String toString() {
        return super.toString() + "@" + color;
    }
}

class Test {
    public static void main(String[] args) {
        Point[] pa = new ColoredPoint[4];
        pa[0] = new ColoredPoint(2, 2, 12);
        pa[1] = new ColoredPoint(4, 5, 24);
        ColoredPoint[] cpa = (ColoredPoint[])pa;
        System.out.print("cpa: {");
        for (int i = 0; i < cpa.length; i++)
            System.out.print((i == 0 ? " " : ", ") + cpa[i]);
        System.out.println(" }");
    }
}
```



```
}
```

This example compiles without errors and produces the output:

```
cpa: { (2,2)@12, (4,5)@24, null, null }
```

The following example uses casts to compile, but it throws exceptions at run time, because the types are incompatible:

```
public class Point { int x, y; }

public interface Colorable { void setColor(int color); }

public class ColoredPoint extends Point implements Colorable
{
    int color;

    public void setColor(int color) { this.color = color; }
}

class Test {
    public static void main(String[] args) {
        Point[] pa = new Point[100];

        // The following line will throw a ClassCastException:
        ColoredPoint[] cpa = (ColoredPoint[])pa;

        System.out.println(cpa[0]);

        int[] shortvec = new int[2];

        Object o = shortvec;

        // The following line will throw a ClassCastException:
        Colorable c = (Colorable)o;

        c.setColor(0);
    }
}
```



```
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.6 Numeric Promotions

Numeric promotion is applied to the operands of an arithmetic operator. Numeric promotion contexts allow the use of an identity conversion (§5.1.1) or a widening primitive conversion (§5.1.2).

Numeric promotions are used to convert the operands of a numeric operator to a common type so that an operation can be performed. The two kinds of numeric promotion are unary numeric promotion (§5.6.1) and binary numeric promotion (§5.6.2). The analogous conversions in C are called "the usual unary conversions" and "the usual binary conversions."

Numeric promotion is not a general feature of Java, but rather a property of the specific definitions of the built-in operations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.6.1 Unary Numeric Promotion

Some operators apply *unary numeric promotion* to a single operand, which must produce a value of a numeric type:

- If the operand is of compile-time type `byte`, `short`, or `char`, unary numeric promotion promotes it to a value of type `int` by a widening conversion (§5.1.2).
- Otherwise, a unary numeric operand remains as is and is not converted.

Unary numeric promotion is performed on expressions in the following situations:

- The dimension expression in array creations (§15.9)
- The index expression in array access expressions (§15.12)
- Operands of the unary operators plus + (§15.14.3) and minus - (§15.14.4)
- The operand of the bitwise complement operator ~ (§15.14.5)
- Each operand, separately, of the shift operators `>>`, `>>>`, and `<<` (§15.18), so that a `long` shift distance (right operand) does not promote the value being shifted (left operand) to `long`

Here is a test program that includes examples of unary numeric promotion:

```
class Test {
    public static void main(String[] args) {
        byte b = 2;
        int a[] = new int[b];
        dimension expression promotion
        char c = '\u0001';
        a[c] = 1;
        expression promotion
        a[0] = -c;
        promotion
        System.out.println("a: " + a[0] + "," + a[1]);

        b = -1;
        int i = ~b;
        complement promotion
        System.out.println("~0x" + Integer.toHexString(b)
                           + "==0x" +
        Integer.toHexString(i));

        i = b << 4L;
        promotion (left operand)
        System.out.println("0x" + Integer.toHexString(b)
                           + "<<4L==0x" + Integer.toHexString(i));
    }
}
```

This test program produces the output:

```
a: -1,1
```



```
~0xffffffff==0x0  
0xffffffff<<4L==0xffffffff0
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


5.6.2 Binary Numeric Promotion

When an operator applies *binary numeric promotion* to a pair of operands, each of which must denote a value of a numeric type, the following rules apply, in order, using widening conversion (§5.1.2) to convert operands as necessary:

- If either operand is of type `double`, the other is converted to `double`.
- Otherwise, if either operand is of type `float`, the other is converted to `float`.
- Otherwise, if either operand is of type `long`, the other is converted to `long`.
- Otherwise, both operands are converted to type `int`.

Binary numeric promotion is performed on the operands of certain operators:

- The multiplicative operators `*`, `/` and `%` (§15.16)
- The addition and subtraction operators for numeric types `+` and `-` (§15.17.2)
- The numerical comparison operators `<`, `<=`, `>`, and `>=` (§15.19.1)
- The numerical equality operators `==` and `!=` (§15.20.1)
- The integer bitwise operators `&`, `^`, and `|` (§15.21.1)
- In certain cases, the conditional operator `?:` (§15.24)

An example of binary numeric promotion appears above in §5.1. Here is another:

```
class Test {
    public static void main(String[] args) {
        int i = 0;
        float f = 1.0f;
        double d = 2.0;

        // First i*f promoted to float*float, then
        // float==double is promoted to double==double:
        if (i * f == d)
            System.out.println("oops");

        // A char&byte is promoted to int&int:
        byte b = 0x1f;
        char c = 'G';
        int control = c & b;
        System.out.println(Integer.toHexString(control));

        // A int:float promoted to float:float:
        f = (b==0) ? f : 4.0f;
        System.out.println(1.0/f);
    }
}
```


which produces the output:

```
7
0.25
```

The example converts the ASCII character `G` to the ASCII control-G (BEL), by masking off all but the low 5 bits of the character. The `7` is the numeric value of this control character.

O suns! O grass of graves! O perpetual transfers and promotions!
--Walt Whitman, *Walt Whitman (1855)* in *Leaves of Grass*

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Names

*The Tao that can be told is not the eternal Tao;
The name that can be named is not the eternal name.
The Nameless is the origin of Heaven and Earth;
The Named is the mother of all things.*
--Lao-Tsu (c. 6th century BC)

Names are used to refer to entities declared in a Java program. A declared entity (§6.1) is a package, class type, interface type, member (field or method) of a reference type, parameter (to a method, constructor, or exception handler), or local variable.

Names in Java programs are either simple, consisting of a single identifier, or qualified, consisting of a sequence of identifiers separated by "." tokens (§6.2).

Every name introduced by a declaration has a *scope* (§6.3), which is the part of the Java program text within which the declared entity can be referred to by a simple name.

Packages and reference types (that is, class types, interface types, and array types) have members (§6.4). A member can be referred to using a qualified name *N.x*, where *N* is a simple or qualified name and *x* is an identifier. If *N* names a package, then *x* is a member of that package, which is either a class or interface type or a subpackage. If *N* names a reference type or a variable of a reference type, then *x* names a member of that type, which is either a field or a method.

In determining the meaning of a name (§6.5), Java uses the context of the occurrence to disambiguate among packages, types, variables, and methods with the same name.

Access control (§6.6) can be specified in a class, interface, method, or field declaration to control when access to a member is allowed. Access is a different concept from scope; access specifies the part of the Java program text within which the declared entity can be referred to by a qualified name, a field access expression (§15.10), or a method invocation expression (§15.11) in which the method is not specified by a simple name. The default access is that a member can be accessed anywhere within the package that contains its declaration; other possibilities are `public`, `protected`, and `private`.

Fully qualified names (§6.7) and naming conventions (§6.8) are also discussed in this chapter.

The name of a field, parameter, or local variable may be used as an expression (§15.13.1). The name of a method may appear in an expression only as part of a method invocation expression (§15.11). The name of a class or interface type may appear in an expression only as part of a class instance creation expression (§15.8), an array creation expression (§15.9), a cast expression (§15.15), or an `instanceof` expression (§15.19.2), or as part of a qualified name for a field or method. The name of a package may appear in an expression only as part of a qualified name for a class or interface type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.1 Declarations

A *declaration* introduces an entity into a Java program and includes an identifier ([§3.8](#)) that can be used in a name to refer to this entity. A declared entity is one of the following:

- A package, declared in a `package` declaration ([§7.4](#))
- An imported type, declared in a single-type-import declaration ([§7.5.1](#)) or a type-import-on-demand declaration ([§7.5.2](#))
- A class, declared in a class type declaration ([§8.1](#))
- An interface, declared in an interface type declaration ([§9.1](#))
- A member of a reference type ([§8.2](#), [§9.2](#), [§10.7](#)), one of the following:
 - A field, one of the following:
 - A field declared in a class type ([§8.3](#))
 - A constant field declared in an interface type ([§9.3](#))
 - The field `length`, which is implicitly a member of every array type ([§10.7](#))
 - A method, one of the following:
 - A method (`abstract` or otherwise) declared in a class type ([§8.4](#))
 - A method (always `abstract`) declared in an interface type ([§9.4](#))
- A parameter, one of the following:
 - A parameter of a method or constructor of a class ([§8.4.1](#), [§8.6.1](#))
 - A parameter of an `abstract` method of an interface ([§9.4](#))
 - A parameter of an exception handler declared in a `catch` clause of a `try` statement ([§14.18](#))
- A local variable, one of the following:
 - A local variable declared in a block ([§14.3](#))
 - A local variable declared in a `for` statement ([§14.12](#))

Constructors ([§8.6](#)) are also introduced by declarations, but use the name of the class in which they are declared rather than introducing a new name.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.2 Names

A *name* is used to refer to an entity declared in a Java program.

There are two forms of names: simple names and qualified names. A *simple name* is a single identifier. A *qualified name* consists of a name, a "." token, and an identifier.

In determining the meaning of a name (§6.5), the Java language takes into account the context in which the name appears. It distinguishes among contexts where a name must denote (refer to) a package (§6.5.3), a type (§6.5.4), a variable or value in an expression (§6.5.5), or a method (§6.5.6).

Not all identifiers in Java programs are a part of a name. Identifiers are also used in the following situations:

- In declarations (§6.1), where an identifier may occur to specify the name by which the declared entity will be known
- In field access expressions (§15.10), where an identifier occurs after a "." token to indicate a member of an object that is the value of an expression or the keyword `super` that appears before the "." token
- In some method invocation expressions (§15.11), where an identifier may occur after a "." token and before a "(" token to indicate a method to be invoked for an object that is the value of an expression or the keyword `super` that appears before the "." token
- As labels in labeled statements (§14.6) and in `break` (§14.13) and `continue` (§14.14) statements that refer to statement labels

In the example:

```
class Test {
    public static void main(String[] args) {
        Class c = System.out.getClass();
        System.out.println(c.toString().length() +
                           args[0].length() +
                           args.length);
    }
}
```

the identifiers `Test`, `main`, and the first occurrences of `args` and `c` are not names; rather, they are used in declarations to specify the names of the declared entities. The names `String`, `Class`, `System.out.getClass`, `System.out.println`, `c.toString`, `args`, and `args.length` appear in the example. The first occurrence of `length` is not a name, but rather an identifier appearing in a method invocation expression (§15.11). The second occurrence of `length` is not a name, but rather an identifier appearing in a method invocation expression (§15.11).

The identifiers used in labeled statements and their associated `break` and `continue` statements are completely separate from those used in declarations. Thus, the following code is valid:

```
class TestString {
```



```

char[] value;

int offset, count;

int indexOf(TestString str, int fromIndex) {
    char[] v1 = value, v2 = str.value;
    int max = offset + (count - str.count);
    int start = offset + ((fromIndex < 0) ? 0 : fromIndex);
i:
    for (int i = start; i <= max; i++)

        {
            int n = str.count, j = i, k = str.offset;
            while (n-- != 0) {
                if (v1[j++] != v2[k++])
                    continue i;
            }
            return i - offset;
        }
    return -1;
}
}

```

This code was taken from a version of the class `String` and its method `indexOf` ([§20.12.26](#)), where the label was originally called `test`. Changing the label to have the same name as the local variable `i` does not hide the label in the scope of the declaration of `i`. The identifier `max` could also have been used as the statement label; the label would not hide the local variable `max` within the labeled statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.3 Scope of a Simple Name

The *scope* of a declaration is the region of the program within which the entity declared by the declaration can be referred to using a simple name:

- The scope of a package, as introduced by a `package` declaration, is determined by the host system (§7.4.3). All Java code is within the scope of the standard package named `java`, so the package `java` can always be referred to by Java code.
- The scope of a type imported by a single-type-import declaration (§7.5.1) or type-import-on-demand declaration (§7.5.2) is all the class and interface type declarations (§7.6) in the compilation unit in which the import declaration appears.
- The scope of a type introduced by a class type declaration (§8.1.1) or interface type declaration (§9.1.1) is the declarations of all class and interface types in all the compilation units (§7.3) of the package in which it is declared.
- The scope of a member declared in or inherited by a class type (§8.2) or interface type (§9.2) is the entire declaration of the class or interface type. The declaration of a member needs to appear before it is used only when the use is in a field initialization expression (§8.3.2, §12.4.2, §12.5). This means that a compile-time error results from the test program:

```
class Test {
    int i = j;                // compile-time error: incorrect
forward reference
    int j = 1;
}
```

whereas the following example compiles without error:

```
class Test {
    Test() { k = 2; }
    int j = 1;
    int i = j;
    int k;
}
```

even though the constructor (§8.6) for `Test` refers to the field `k` that is declared three lines later.

- The scope of a parameter of a method (§8.4.1) is the entire body of the method.
- The scope of a parameter of a constructor (§8.6.1) is the entire body of the constructor.
- The scope of a local variable declaration in a block (§14.3.2) is the rest of the block in which the declaration appears, starting with its own initializer (§14.3) and including any further declarators to the right in the local variable declaration statement.
- The scope of a local variable declared in the *ForInit* part of a `for` statement (§14.12) includes all of the following:
 - Its own initializer
 - Any further declarators to the right in the *ForInit* part of the `for` statement
 - The *Expression* and *ForUpdate* parts of the `for` statement

- The contained *Statement*
- The scope of a parameter of an exception handler that is declared in a `catch` clause of a `try` statement (§14.18) is the entire block associated with the `catch`.

These rules imply that declarations of class and interface types need not appear before uses of the types.

In the example:

```
package points;
```

```
class Point {
    int x, y;
    PointList list;
    Point next;
}
```

```
class PointList {
    Point first;
}
```

the use of `PointList` in class `Point` is correct, because the scope of the class type name `PointList` includes both class `Point` and class `PointList`, as well as any other type declarations in other compilation units of package `points`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.3.1 Hiding Names

Some declarations may be hidden ([§6.3.1](#)) in part of their scope by another declaration of the same name, in which case a simple name cannot be used to refer to the declared entity.

The example:

```
class Test {
    static int x = 1;
    public static void main(String[] args) {
        int x = 0;
        System.out.print("x=" + x);
        System.out.println(", Test.x=" + Test.x);
    }
}
```

produces the output:

```
x=0, Test.x=1
```

This example declares:

- a class `Test`
- a class (static) variable `x` that is a member of the class `Test`
- a class method `main` that is a member of the class `Test`
- a parameter `args` of the `main` method
- a local variable `x` of the `main` method

Since the scope of a class variable includes the entire body of the class ([§8.2](#)) the class variable `x` would normally be available throughout the entire body of the method `main`. In this example, however, the class variable `x` is hidden within the body of the method `main` by the declaration of the local variable `x`.

A local variable has as its scope the rest of the block in which it is declared ([§14.3.2](#)); in this case this is the rest of the body of the `main` method, namely its initializer "0" and the invocations of `print` and `println`.

This means that:

- The expression "x" in the invocation of `print` refers to (denotes) the value of the local variable `x`.
- The invocation of `println` uses a qualified name ([§6.6](#)) `Test.x`, which uses the class type name `Test` to access the class variable `x`, because the declaration of `Test.x` is hidden at this point and cannot be referred to by its simple name.

If the standard naming conventions ([§6.8](#)) are followed, then hiding that would make the identification of separate naming contexts matter should be rare. The following contrived example involves hiding because it does not follow the standard naming conventions:


```

class Point { int x, y; }

class Test {

    static Point Point(int x, int y) {
        Point p = new Point();
        p.x = x; p.y = y;
        return p;
    }

    public static void main(String[] args) {
        int Point;
        Point[] pa = new Point[2];
        for (Point = 0; Point < 2; Point++) {
            pa[Point] = new Point();
            pa[Point].x = Point;
            pa[Point].y = Point;
        }
        System.out.println(pa[0].x + "," + pa[0].y);
        System.out.println(pa[1].x + "," + pa[1].y);
        Point p = Point(3, 4);
        System.out.println(p.x + "," + p.y);
    }
}

```

This compiles without error and executes to produce the output:

```

0,0
1,1
3,4

```

Within the body of `main`, the lookups of `Point` find different declarations depending on the context of the use:

- In the expression "`new Point[2]`", the two occurrences of the class instance creation expression "`new Point()`", and at the start of three different local variable declaration statements, the `Point` is a *TypeName* (§6.5.4) and denotes the class type `Point` in each case.
- In the method invocation expression "`Point(3, 4)`" the occurrence of `Point` is a *MethodName* (§6.5.6) and denotes the class (static) method `Point`.
- All other names are *ExpressionNames* (§6.5.5) and refer to the local variable `Point`.

The example:

```

import java.util.*;

```



```
class Vector {  
    int val[] = { 1 , 2 };  
}  
  
class Test {  
    public static void main(String[] args) {  
        Vector v = new Vector();  
        System.out.println(v.val[0]);  
    }  
}
```

compiles and prints:

1

using the class `Vector` declared here in preference to class `java.util.Vector` that might be imported on demand.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

6.4 Members and Inheritance

Packages and reference types have *members*. The members of a package (§7) are subpackages (§7.1) and all the class (§8) and interface (§9) types declared in all the compilation units (§7.3) of the package. The members of a reference type (§4.3) are fields (§8.3, §9.3, §10.7) and methods (§8.4, §9.4). Members are either declared in the type, or *inherited* because they are accessible members of a superclass or superinterface which are neither hidden nor overridden (§8.4.6).

This section provides an overview of the members of packages and reference types here, as background for the discussion of qualified names and the determination of the meaning of names. For a complete description of membership, see §7.1, §8.2, §9.2, and §10.7.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.4.1 The Members of a Package

A member of a package (§7) is a subpackage (§7.1), or a class (§8) or interface (§9) type declared in a compilation unit (§7.3) of the package.

In general, the subpackages of a package are determined by the host system (§7.2). However, the standard package `java` always includes the subpackages `lang`, `util`, `io`, and `net` and may include other subpackages. No two distinct members of the same package may have the same simple name (§7.1), but members of different packages may have the same simple name. For example, it is possible to declare a package:

```
package vector;  
public class Vector { Object[] vec; }
```

that has as a member a public class named `Vector`, even though the standard package `java.util` also declares a class named `Vector`. These two class types are different, reflected by the fact that they have different fully qualified names (§6.7). The fully qualified name of this example `Vector` is `vector.Vector`, whereas `java.util.Vector` is the fully qualified name of the standard `Vector` class. Because the package `vector` contains a class named `Vector`, it cannot also have a subpackage named `Vector`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.4.2 The Members of a Class Type

The members of a class type (§8.2) are fields and methods. The members of a class type are all of the following:

- Members inherited from its direct superclass (§8.1.3), if it has one (the class `Object` has no direct superclass)
- Members inherited from any direct superinterfaces (§8.1.4)
- Members declared in the body of the class (§8.1.5)

Constructors (§8.6) are not members.

There is no restriction against a field and a method of a class type having the same simple name.

A class may have two or more fields with the same simple name if they are declared in different interfaces and inherited. An attempt to refer to any of the fields by its simple name results in a compile-time error (§6.5.6.2, §8.2).

In the example:

```
interface Colors {
    int WHITE = 0, BLACK = 1;
}

interface Separates {
    int CYAN = 0, MAGENTA = 1, YELLOW = 2, BLACK = 3;
}

class Test implements Colors, Separates {
    public static void main(String[] args) {
        System.out.println(BLACK); // compile-time error: ambiguous
    }
}
```

the name `BLACK` in the method `main` is ambiguous, because class `Test` has two members named `BLACK`, one inherited from `Colors` and one from `Separates`.

A class type may have two or more methods with the same simple name if the methods have different signatures (§8.4.2), that is, if they have different numbers of parameters or different parameter types in at least one parameter position. Such a method member name is said to be *overloaded*.

A class type may contain a declaration for a method with the same name and the same signature as a method that would otherwise be inherited from a superclass or superinterface. In this case, the method of the superclass or superinterface is not inherited. If the method not inherited is *abstract*, then the new declaration is said to *implement* it; if the method not inherited is not *abstract*, then the new declaration is said to *override* it.

In the example:

```
class Point {
    float x, y;
```



```

void move(int dx, int dy) { x += dx; y += dy; }
void move(float dx, float dy) { x += dx; y += dy; }
public String toString() { return "("+x+","+y+")"; }
}

```

the class `Point` has two members that are methods with the same name, `move`. The overloaded `move` method of class `Point` chosen for any particular method invocation is determined at compile time by the overloading resolution procedure given in [§15.11](#).

In this example, the members of the class `Point` are the `float` instance variables `x` and `y` declared in `Point`, the two declared `move` methods, the declared `toString` method, and the members that `Point` inherits from its implicit direct superclass `Object` ([§4.3.2](#)), such as the method `hashCode` ([§20.1.4](#)). Note that `Point` does not inherit the `toString` method ([§20.1.2](#)) of class `Object` because that method is overridden by the declaration of the `toString` method in class `Point`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.4.3 The Members of an Interface Type

The members of an interface type (§9.2) are fields and methods. The members of an interface are all of the following:

- Members inherited from any direct superinterfaces (§9.1.3)
- Members declared in the body of the interface (§9.1.4)

An interface may have two or more fields with the same simple name if they are declared in different interfaces and inherited. An attempt to refer to any such field by its simple name results in a compile-time error (§6.5.5.1, §9.2).

In the example:

```
interface Colors {
    int WHITE = 0, BLACK = 1;
}

interface Separates {
    int CYAN = 0, MAGENTA = 1, YELLOW = 2, BLACK = 3;
}

interface ColorsAndSeparates extends Colors, Separates {

    int DEFAULT = BLACK;
    // compile-time error: ambiguous

}
```

the members of the interface `ColorsAndSeparates` include those members inherited from `Colors` and those inherited from `Separates`, namely `WHITE`, `BLACK` (first of two), `CYAN`, `MAGENTA`, `YELLOW`, and `BLACK` (second of two). The member name `BLACK` is ambiguous in the interface `ColorsAndSeparates`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

6.4.4 The Members of an Array Type

The members of an array type ([§10.7](#)) are all of the following:

- Members inherited from its implicit superclass `Object` ([§4.3.2](#), [§20.1](#))
- The field `length`, which is a constant (`final`) field of every array; its type is `int` and it contains the number of components of the array

The example:

```
class Test {
    public static void main(String[] args) {
        int[] ia = new int[3];
        int[] ib = new int[6];
        System.out.println(ia.getClass() == ib.getClass());
        System.out.println("ia has length=" + ia.length);
    }
}
```

produces the output:

```
true
ia has length=3
```

This example uses the method `getClass` inherited from class `Object` and the field `length`. The result of the comparison of the `Class` objects in the second `println` demonstrates that all arrays whose components are of type `int` are instances of the same array type, which is `int[]`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

6.5 Determining the Meaning of a Name

The meaning of a name in Java depends on the context in which it is used. The determination of the meaning of a name requires three steps. First, context causes a name syntactically to fall into one of five categories: *PackageName*, *TypeName*, *ExpressionName*, *MethodName*, or *AmbiguousName*. Second, a name that is initially classified by its context as an *AmbiguousName* is then reclassified by certain scoping rules to be a *PackageName*, *TypeName*, or *ExpressionName*. Third, the resulting category then dictates the final determination of the meaning of the name (or a compilation error if the name has no meaning).

PackageName:

Identifier

PackageName . Identifier

TypeName:

Identifier

PackageName . Identifier

ExpressionName:

Identifier

AmbiguousName . Identifier

MethodName:

Identifier

AmbiguousName . Identifier

AmbiguousName:

Identifier

AmbiguousName . Identifier

Java's use of context helps to minimize name conflicts between entities of different kinds. Such conflicts will be rare if the naming conventions described in [§6.8](#) are followed. Nevertheless, conflicts may arise unintentionally as types developed by different programmers or different organizations evolve. For example, types, methods, and fields may have the same name. Java never has trouble distinguishing between a method and a field with the same name, since the context of a use always tells whether a method or a field is intended.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.5.1 Syntactic Classification of a Name According to Context

A name is syntactically classified as a *PackageName* in these contexts:

- In a package declaration ([§7.4](#))
- In a type-import-on-demand declaration ([§7.5.2](#))
- To the left of the "." in a qualified *PackageName*
- To the left of the "." in a qualified *TypeName*

A name is syntactically classified as a *TypeName* in these contexts:

- In a single-type-import declaration ([§7.5.1](#))
- In an `extends` clause in a class declaration ([§8.1.3](#))
- In an `implements` clause in a class declaration ([§8.1.4](#))
- In an `extends` clause in an interface declaration ([§9.1.3](#))
- As a *Type* (or the part of a *Type* that remains after all brackets are deleted) in any of the following contexts:
 - In a field declaration ([§8.3](#), [§9.3](#))
 - As the result type of a method ([§8.4](#), [§9.4](#))
 - As the type of a formal parameter of a method or constructor ([§8.4.1](#), [§8.6.1](#), [§9.4](#))
 - As the type of an exception that can be thrown by a method or constructor ([§8.4.4](#), [§8.6.4](#), [§9.4](#))
 - As the type of a local variable ([§14.3](#))
 - As the type of an exception parameter in a `catch` clause of a `try` statement ([§14.18](#))
 - As the class type of an instance that is to be created in a class instance creation expression ([§15.8](#))
 - As the element type of an array to be created in an array creation expression ([§15.9](#))
 - As the type mentioned in the cast operator of a cast expression ([§15.15](#))
 - As the type that follows the `instanceof` relational operator ([§15.19.2](#))

A name is syntactically classified as an *ExpressionName* in these contexts:

- As the array reference expression in an array access expression ([§15.12](#))
- As a *PostfixExpression* ([§15.13](#))
- As the left-hand operand of an assignment operator ([§15.25](#))

A name is syntactically classified as a *MethodName* in this context:

- Before the "(" in a method invocation expression ([§15.11](#))

A name is syntactically classified as an *AmbiguousName* in these contexts:

- To the left of the "." in a qualified *ExpressionName*
- To the left of the "." in a qualified *MethodName*
- To the left of the "." in a qualified *AmbiguousName*


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.5.2 Reclassification of Contextually Ambiguous Names

An *AmbiguousName* is then reclassified as follows:

- If the *AmbiguousName* is a simple name, consisting of a single *Identifier*:
 - If the *Identifier* appears within the scope (§6.3) of a local variable declaration (§14.3) or parameter declaration (§8.4.1, §8.6.1, §14.18) with that name, then the *AmbiguousName* is reclassified as an *ExpressionName*.
 - Otherwise, consider the class or interface *C* within whose declaration the *Identifier* occurs. If *C* has one or more fields with that name, which may be either declared within it or inherited, then the *AmbiguousName* is reclassified as an *ExpressionName*.
 - Otherwise, if a type of that name is declared in the compilation unit (§7.3) containing the *Identifier*, either by a single-type-import declaration (§7.5.1) or by a class or interface type declaration (§7.6), then the *AmbiguousName* is reclassified as a *TypeName*.
 - Otherwise, if a type of that name is declared in another compilation unit (§7.3) of the package (§7.1) of the compilation unit containing the *Identifier*, then the *AmbiguousName* is reclassified as a *TypeName*.
 - Otherwise, if a type of that name is declared by exactly one type-import-on-demand declaration (§7.5.2) of the compilation unit containing the *Identifier*, then the *AmbiguousName* is reclassified as a *TypeName*.
 - Otherwise, if a type of that name is declared by more than one type-import-on-demand declaration of the compilation unit containing the *Identifier*, then a compile-time error results.
 - Otherwise, the *AmbiguousName* is reclassified as a *PackageName*. A later step determines whether or not a package of that name actually exists.
- If the *AmbiguousName* is a qualified name, consisting of a name, a ".", and an *Identifier*, then the name to the left of the "." is first reclassified, for it is itself an *AmbiguousName*. There is then a choice:
 - If the name to the left of the "." is reclassified as a *PackageName*, then there is a further choice:
 - If there is a package whose name is the name to the left of the "." and that package contains a declaration of a type whose name is the same as the *Identifier*, then this *AmbiguousName* is reclassified as a *TypeName*.
 - Otherwise, this *AmbiguousName* is reclassified as a *PackageName*. A later step determines whether or not a package of that name actually exists.
 - If the name to the left of the "." is reclassified as a *TypeName*, then this *AmbiguousName* is reclassified as an *ExpressionName*. If the name to the left of the "." is reclassified as an *ExpressionName*, then this *AmbiguousName* is reclassified as an *ExpressionName*.

As an example, consider the following contrived "library code":

```
package ORG.rpgpoet;
```

```
import java.util.Random;
```



```
interface Music { Random[] wizards = new Random[4]; }
```

and then consider this example code in another package:

```
package bazola;
```

```
class Gabriel {  
    static int n = ORG.rpgpoet.Music.wizards.length;  
}
```

First of all, the name `ORG.rpgpoet.Music.wizards.length` is classified as an *ExpressionName* because it functions as a *PostfixExpression*. Therefore, each of the names:

```
ORG.rpgpoet.Music.wizards  
ORG.rpgpoet.Music  
ORG.rpgpoet  
ORG
```

is initially classified as an *AmbiguousName*. These are then reclassified:

- Assuming that there is no class or interface named `ORG` in any other compilation unit of package `bazola`, then the simple name `ORG` is reclassified as a *PackageName*.
- Next, assuming that there is no class or interface named `rpgpoet` in any compilation unit of package `ORG` (and we know that there is no such class or interface because package `ORG` has a subpackage named `rpgpoet`), the qualified name `ORG.rpgpoet` is reclassified as a *PackageName*.
- Next, because package `ORG.rpgpoet` has an interface type named `Music`, the qualified name `ORG.rpgpoet.Music` is reclassified as a *TypeName*.
- Finally, because the name `ORG.rpgpoet.Music` is a *TypeName*, the qualified name `ORG.rpgpoet.Music.wizards` is reclassified as an *ExpressionName*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.5.3 Meaning of Package Names

The meaning of a name classified as a *PackageName* is determined as follows.

6.5.3.1 Simple Package Names

If a package name consists of a single *Identifier*, then this identifier denotes a top-level package named by that identifier. If no package of that name is accessible, as determined by the host system (§7.4.3), then a compile-time error occurs.

6.5.3.2 Qualified Package Names

If a package name is of the form *Q . Id*, then *Q* must also be a package name. The package name *Q . Id* names a package that is the member named *Id* within the package named by *Q*. If *Q* does not name an accessible package or *Id* does not name an accessible subpackage of that package, then a compile-time error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

6.5.4 Meaning of Type Names

The meaning of a name classified as a *TypeName* is determined as follows.

6.5.4.1 Simple Type Names

If a type name consists of a single *Identifier*, then the identifier must occur in the scope of a declaration of a type with this name, or a compile-time error occurs. It is possible that the identifier occurs within the scope of more than one type with that name, in which case the type denoted by the name is determined as follows:

- If a type with that name is declared in the current compilation unit (§7.3), either by a single-type-import declaration (§7.5.1) or by a declaration of a class or interface type (§7.6), then the simple type name denotes that type.
- Otherwise, if a type with that name is declared in another compilation unit (§7.3) of the package (§7.1) containing the identifier, then the identifier denotes that type. Note that, in systems that store compilation units in a file system, such a compilation unit must have a file name that is the name of the type (§7.6).
- Otherwise, if a type of that name is declared by exactly one type-import-on-demand declaration (§7.5.2) of the compilation unit containing the identifier, then the simple type name denotes that type.
- Otherwise, if a type of that name is declared by more than one type-import-on-demand declaration of the compilation unit, then the name is ambiguous as a type name; a compile-time error occurs.
- Otherwise, the name is undefined as a type name; a compile-time error occurs.

This order for considering type declarations is designed to choose the most explicit of two or more applicable type declarations.

6.5.4.2 Qualified Type Names

If a type name is of the form *Q.Id*, then *Q* must be a package name. The type name *Q.Id* names a type that is the member named *Id* within the package named by *Q*. If *Q* does not name an accessible package, or *Id* does not name a type within that package, or the type named *Id* within that package is not accessible (§6.6), then a compile-time error occurs.

The example:

```
package wnj.test;
```

```
class Test {
    public static void main(String[] args) {
        java.util.Date date =
            new java.util.Date(System.currentTimeMillis());
        System.out.println(date.toLocaleString());
    }
}
```


produced the following output the first time it was run:

```
Sun Jan 21 22:56:29 1996
```

In this example:

- The name `wnj.test` must name a package on the host system. It is resolved by first looking for the package `wnj`, using the procedure described in [§6.5.3.1](#), and then making sure that the subpackage `test` of this package is accessible.
- The name `java.util.Date` ([§21.3](#)) must denote a type, so we first use the procedure recursively to determine if `java.util` is an accessible package, which it is, and then look to see if the type `Date` is accessible in this package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.5.5 Meaning of Expression Names

The meaning of a name classified as an *ExpressionName* is determined as follows.

6.5.5.1 Simple Expression Names

If an expression name consists of a single *Identifier*, then:

- If the *Identifier* appears within the scope (§6.3) of a local variable declaration (§14.3) or parameter declaration (§8.4.1, §8.6.1, §14.18) with that name, then the expression name denotes a variable, that is, that local variable or parameter. Local variables and parameters are never hidden (§6.3, §6.3.1, §14.3), so there is necessarily at most one such local variable or parameter. The type of the expression name is the declared type of the local variable or parameter.
- Otherwise, if the *Identifier* appears within a class declaration (§8):
 - If there is not exactly one member of that class (§8.2) that is a field with that name, then a compile-time error results.
 - Otherwise, if the single member field with that name is declared `final` (§8.3.1.2), then the expression name denotes the value of the field. The type of the expression name is the declared type of the field. If the *Identifier* appears in a context that requires a variable and not a value, then a compile-time error occurs.
 - Otherwise, the expression name denotes a variable, the single member field with that name. The type of the expression name is the field's declared type.

If the field is an instance variable (§8.3.1.1), the expression name must appear within the declaration of an instance method (§8.4), constructor (§8.6), or instance variable initializer (§8.3.2.2). If it appears within a `static` method (§8.4.3.2), static initializer (§8.5), or initializer for a `static` variable (§8.3.1.1, §12.4.2), then a compile-time error occurs.

- Otherwise, the identifier appears within an interface declaration (§9):
 - If there is not exactly one member of that interface (§9.2) that is a field with that name, then a compile-time error results.
 - Otherwise, the expression name denotes the value of the single member field of that name. The type of the expression name is the declared type of the field. If the *Identifier* appears in a context that requires a variable and not a value, then a compile-time error occurs.

In the example:

```
class Test {  
  
    static int v;  
  
    static final int f = 3;  
}
```



```

    public static void main(String[] args) {
        int i;
        i = 1;
        v = 2;
        f = 33;
        // compile-time error
        System.out.println(i + " " + v + " " + f);
    }
}

```

the names used as the left-hand-sides in the assignments to `i`, `v`, and `f` denote the local variable `i`, the field `v`, and the value of `f` (not the variable `f`, because `f` is a `final` variable). The example therefore produces an error at compile time because the last assignment does not have a variable as its left-hand side. If the erroneous assignment is removed, the modified code can be compiled and it will produce the output:

```
1 2 3
```

6.5.5.2 Qualified Expression Names

If an expression name is of the form `Q.id`, then `Q` has already been classified as a package name, a type name, or an expression name:

- If `Q` is a package name, then a compile-time error occurs.
- If `Q` is a type name that names a class type (§8), then:
 - If there is not exactly one accessible (§6.6) member of the class type that is a field named `id`, then a compile-time error occurs.
 - Otherwise, if the single accessible member field is not a class variable (that is, it is not declared `static`), then a compile-time error occurs.
 - Otherwise, if the class variable is declared `final`, then `Q.id` denotes the value of the class variable. The type of the expression `Q.id` is the declared type of the class variable. If `Q.id` appears in a context that requires a variable and not a value, then a compile-time error occurs.
 - Otherwise, `Q.id` denotes the class variable. The type of the expression `Q.id` is the declared type of the class variable.
- If `Q` is a type name that names an interface type (§9), then:
 - If there is not exactly one accessible (§6.6) member of the interface type that is a field named `id`, then a compile-time error occurs.
 - Otherwise, `Q.id` denotes the value of the field. The type of the expression `Q.id` is the declared type of the field. If `Q.id` appears in a context that requires a variable and not a value, then a compile-time error occurs.
- If `Q` is an expression name, let `T` be the type of the expression `Q`:
 - If `T` is not a reference type, a compile-time error occurs.
 - If there is not exactly one accessible (§6.6) member of the type `T` that is a field named `id`, then a compile-time error occurs.
 - Otherwise, if this field is any of the following:

- A field of an interface type
- A `final` field of a class type (which may be either a class variable or an instance variable)
- The `final` field `length` of an array type then `Q.l` denotes the value of the field. The type of the expression `Q.l` is the declared type of the field. If `Q.l` appears in a context that requires a variable and not a value, then a compile-time error occurs.
- Otherwise, `Q.l` denotes a variable, the field `l` of class `T`, which may be either a class variable or an instance variable. The type of the expression `Q.l` is the declared type of the field

The example:

```
class Point {
    int x, y;
    static int nPoints;
}

class Test {
    public static void main(String[] args) {
        int i = 0;
        i.x++; //
compile-time error
        Point p = new Point();
        p.nPoints(); //
compile-time error
    }
}
```

encounters two compile-time errors, because the `int` variable `i` has no members, and because `nPoints` is not a method of class `Point`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.5.6 Meaning of Method Names

A *MethodName* can appear only in a method invocation expression (§15.11). The meaning of a name classified as a *MethodName* is determined as follows.

6.5.6.1 Simple Method Names

If a method name consists of a single *Identifier*, then *Identifier* is the method name to be used for method invocation. The *Identifier* must name at least one method of the class or interface within whose declaration the *Identifier* appears. See §15.11 for further discussion of the interpretation of simple method names in method invocation expressions.

6.5.6.2 Qualified Method Names

If a method name is of the form *Q.Id*, then *Q* has already been classified as a package name, a type name, or an expression name. If *Q* is a package name, then a compile-time error occurs. Otherwise, *Id* is the method name to be used for method invocation. If *Q* is a type name, then *Id* must name at least one `static` method of the type *Q*. If *Q* is an expression name, then let *T* be the type of the expression *Q*; *Id* must name at least one method of the type *T*. See §15.11 for further discussion of the interpretation of qualified method names in method invocation expressions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6 Qualified Names and Access Control

Qualified names are a means of access to members of packages and reference types; related means of access include field access expressions (§15.10) and method invocation expressions (§15.11). All three are syntactically similar in that a "." token appears, preceded by some indication of a package, type, or expression having a type and followed by an *Identifier* that names a member of the package or type. These are collectively known as constructs for *qualified access*.

Java provides mechanisms for *access control*, to prevent the users of a package or class from depending on unnecessary details of the implementation of that package or class. Access control applies to qualified access and to the invocation of constructors by class instance creation expressions (§15.8), explicit constructor invocations (§8.6.5), and the method `newInstance` of class `Class` (§20.3.6).

If access is permitted, then the accessed entity is said to be *accessible*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.1 Determining Accessibility

- Whether a package is accessible is determined by the host system (§7.2).
- If a class or interface type is declared `public`, then it may be accessed by any Java code that can access the package in which it is declared. If a class or interface type is not declared `public`, then it may be accessed only from within the package in which it is declared.
- A member (field or method) of a reference (class, interface, or array) type or a constructor of a class type is accessible only if the type is accessible and the member or constructor is declared to permit access:
 - If the member or constructor is declared `public`, then access is permitted. All members of interfaces are implicitly `public`.
 - Otherwise, if the member or constructor is declared `protected`, then access is permitted only when one of the following is true:
 - Access to the member or constructor occurs from within the package containing the class in which the `protected` member is declared.
 - Access occurs within a subclass of the class in which the `protected` member is declared, and the access is correct as described in §6.6.2.
- Otherwise, if the member or constructor is declared `private`, then access is permitted only when it occurs from within the class in which it is declared.
- Otherwise, we say there is default access, which is permitted only when the access occurs from within the package in which the type is declared.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.2 Details on protected Access

A `protected` member or constructor of an object may be accessed from outside the package in which it is declared only by code that is responsible for the implementation of that object. Let *C* be the class in which a `protected` member or constructor is declared and let *S* be the subclass of *C* in whose declaration the use of the `protected` member or constructor occurs. Then:

- If an access is of a `protected` member (field or method), let *Id* be its name. Consider then the means of qualified access:
 - If the access is by a field access expression of the form `super.Id`, then the access is permitted.
 - If the access is by a qualified name `Q.Id`, where *Q* is a *TypeName*, then the access is permitted if and only if *Q* is *S* or a subclass of *S*.
 - If the access is by a qualified name `Q.Id`, where *Q* is an *ExpressionName*, then the access is permitted if and only if the type of the expression *Q* is *S* or a subclass of *S*.
 - If the access is by a field access expression `E.Id`, where *E* is a *Primary* expression, or by a method invocation expression `E.Id(...)`, where *E* is a *Primary* expression, then the access is permitted if and only if the type of *E* is *S* or a subclass of *S*.
- Otherwise, if an access is of a `protected` constructor:
 - If the access is by a superclass constructor invocation `super(...)`, then the access is permitted.
 - If the access is by a class instance creation expression `new T(...)`, then the access is not permitted. (A `protected` constructor can be accessed by a class instance creation expression only from within the package in which it is defined.)
 - If the access is by an invocation of the method `newInstance` of class `Class` ([§20.3.6](#)), then the access is not permitted.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.3 An Example of Access Control

For examples of access control, consider the two compilation units:

```
package points;

class PointVec { Point[] vec; }
```

and:

```
package points;

public class Point {
    protected int x, y;
    public void move(int dx, int dy) { x += dx; y += dy; }
    public int getX() { return x; }
    public int getY() { return y; }
}
```

which declare two class types in the package `points`:

- The class type `PointVec` is not `public` and not part of the `public` interface of the package `points`, but rather can be used only by other classes in the package.
- The class type `Point` is declared `public` and is available to other packages. It is part of the `public` interface of the package `points`.
- The methods `move`, `getX`, and `getY` of the class `Point` are declared `public` and so are available to any Java code that uses an object of type `Point`.
- The fields `x` and `y` are declared `protected` and are accessible outside the package `points` only in subclasses of class `Point`, and only when they are fields of objects that are being implemented by the code that is accessing them.

See [§6.6.7](#) for an example of how the `protected` access modifier limits access.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.4 Example: Access to public and Non-public Classes

If a class lacks the `public` modifier, access to the class declaration is limited to the package in which it is declared ([§6.6](#)). In the example:

```
package points;
```

```
public class Point {  
    public int x, y;  
    public void move(int dx, int dy) { x += dx; y += dy; }  
}
```

```
class PointList {  
    Point next, prev;  
}
```

two classes are declared in the compilation unit. The class `Point` is available outside the package `points`, while the class `PointList` is available for access only within the package. Thus a compilation unit in another package can access `points.Point`, either by using its fully qualified name:

```
package pointsUser;
```

```
class Test {  
    public static void main(String[] args) {  
        points.Point p = new points.Point();  
        System.out.println(p.x + " " + p.y);  
    }  
}
```

or by using a single-type-import declaration ([§7.5.1](#)) that mentions the fully qualified name, so that the simple name may be used thereafter:

```
package pointsUser;
```

```
import points.Point;
```



```
class Test {  
    public static void main(String[] args) {  
        Point p = new Point();  
        System.out.println(p.x + " " + p.y);  
    }  
}
```

However, this compilation unit cannot use or import `points.PointList`, which is not declared `public` and is therefore inaccessible outside package `points`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.5 Example: Default-Access Fields, Methods, and Constructors

If none of the access modifiers `public`, `protected`, or `private` are specified, a class member or constructor is accessible throughout the package that contains the declaration of the class in which the class member is declared, but the class member or constructor is not accessible in any other package. If a `public` class has a method or constructor with default access, then this method or constructor is not accessible to or inherited by a subclass declared outside this package.

For example, if we have:

```
package points;
```

```
public class Point {
    public int x, y;
    void move(int dx, int dy) { x += dx; y += dy; }
    public void moveAlso(int dx, int dy) { move(dx, dy); }
}
```

then a subclass in another package may declare an unrelated `move` method, with the same signature ([§8.4.2](#)) and return type, but because the original `move` method is not accessible from package `morepoints` `super` may not be used:

```
package morepoints;
```

```
public class PlusPoint extends points.Point {
    public void move(int dx, int dy) {
        super.move(dx, dy);
        // compile-time error
        moveAlso(dx, dy);
    }
}
```

Because `move` of `Point` is not overridden by `move` in `PlusPoint`, the method `moveAlso` in `Point` never calls the method `move` in `PlusPoint`.

Thus if you delete the `super.move` call from `PlusPoint` and execute the test program:

```
import points.Point;

import morepoints.PlusPoint;
```



```
class Test {  
  
    public static void main(String[] args) {  
        PlusPoint pp = new PlusPoint();  
        pp.move(1, 1);  
    }  
  
}
```

it terminates normally. If `move` of `Point` were overridden by `move` in `PlusPoint`, then this program would recurse infinitely, until a `StackoverflowError` occurred.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.6 Example: public Fields, Methods, and Constructors

A `public` class member or constructor is accessible throughout the package where it is declared and from any other package that has access to the package in which it is declared ([§7.4.4](#)). For example, in the compilation unit:

```
package points;

public class Point {

    int x, y;

    public void move(int dx, int dy) {
        x += dx; y += dy;
        moves++;
    }

    public static int moves = 0;
}
```

the `public class Point` has as `public` members the `move` method and the `moves` field. These `public` members are accessible to any other package that has access to `package points`. The fields `x` and `y` are not `public` and therefore are accessible only from within the `package points`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.7 Example: protected Fields, Methods, and Constructors

Consider this example, where the `point` package declares:

```
package points;

public class Point {
    protected int x, y;

    void warp(threePoint.Point3d a) {
        if (a.z > 0) // compile-time
            error: cannot access a.z
            a.delta(this);
    }
}
```

and the `threePoint` package declares:

```
package threePoint;

import points.Point;

public class Point3d extends Point {
    protected int z;

    public void delta(Point p) {
        p.x += this.x; // compile-time
        error: cannot access p.x
        p.y += this.y; // compile-time
        error: cannot access p.y
    }

    public void delta3d(Point3d q) {
        q.x += this.x;
        q.y += this.y;
        q.z += this.z;
    }
}
```



```
}  
  
}
```

which defines a class `Point3d`. A compile-time error occurs in the method `delta` here: it cannot access the protected members `x` and `y` of its parameter `p`, because while `Point3d` (the class in which the references to fields `x` and `y` occur) is a subclass of `Point` (the class in which `x` and `y` are declared), it is not involved in the implementation of a `Point` (the type of the parameter `p`). The method `delta3d` can access the protected members of its parameter `q`, because the class `Point3d` is a subclass of `Point` and is involved in the implementation of a `Point3d`.

The method `delta` could try to cast (§5.4, §15.15) its parameter to be a `Point3d`, but this cast would fail, causing an exception, if the class of `p` at run time were not `Point3d`.

A compile-time error also occurs in the method `warp`: it cannot access the protected member `z` of its parameter `a`, because while the class `Point` (the class in which the reference to field `z` occurs) is involved in the implementation of a `Point` (the type of the parameter `a`), it is not a subclass of `Point` (the class in which `z` is declared).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.6.8 Example: private Fields, Methods, and Constructors

A `private` class member or constructor is accessible only within the class body in which the member is declared and is not inherited by subclasses. In the example:

```
class Point {  
    Point() { setMasterID(); }  
  
    int x, y;  
    private int ID;  
    private static int masterID = 0;  
  
    private void setMasterID() { ID = masterID++; }  
}
```

the `private` members `ID`, `masterID`, and `setMasterID` may be used only within the body of class `Point`. They may not be accessed by qualified names, field access expressions, or method invocation expressions outside the body of the declaration of `Point`.

See [§8.6.8](#) for an example that uses a `private` constructor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.7 Fully Qualified Names

Every package, class, interface, array type, and primitive type has a fully qualified name. It follows that every type except the null type has a fully qualified name.

- The fully qualified name of a primitive type is the keyword for that primitive type, namely `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, or `double`.
- The fully qualified name of a named package that is not a subpackage of a named package is its simple name.
- The fully qualified name of a named package that is a subpackage of another named package consists of the fully qualified name of the containing package, followed by `"."`, followed by the simple (member) name of the subpackage.
- The fully qualified name of a class or interface that is declared in an unnamed package is the simple name of the class or interface.
- The fully qualified name of a class or interface that is declared in a named package consists of the fully qualified name of the package, followed by `"."`, followed by the simple name of the class or interface.
- The fully qualified name of an array type consists of the fully qualified name of the component type of the array type followed by `"[]"`.

Examples:

- The fully qualified name of the type `long` is `"long"`.
- The fully qualified name of the standard package `java.lang` is `"java.lang"` because it is subpackage `lang` of package `java`.
- The fully qualified name of the class `Object`, which is defined in the package `java.lang`, is `"java.lang.Object"`.
- The fully qualified name of the interface `Enumeration`, which is defined in the package `java.util`, is `"java.util.Enumeration"`.
- The fully qualified name of the type "array of `double`" is `"double[]"`.
- The fully qualified name of the type "array of array of array of array of `String`" is `"java.lang.String[][][][]"`.

In the example:

```
package points;
```

```
class Point { int x, y; }
```

```
class PointVec {  
    Point[] vec;  
}
```

the fully qualified name of the type `Point` is `"points.Point"`; the fully qualified name of the type `PointVec` is `"points.PointVec"`; and the fully qualified name of the type of the field `vec` of class `PointVec` is `"points.Point[]"`.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8 Naming Conventions

The Java system and standard classes attempt to use, whenever possible, names chosen according to the conventions presented here. These conventions help to make code more readable and avoid certain kinds of name conflicts.

We recommend these conventions for use in all Java programs. However, these conventions should not be followed slavishly if long-held conventional usage dictates otherwise. So, for example, the `sin` and `cos` methods of the class `java.lang.Math` have mathematically conventional names, even though these method names flout Java convention because they are short and are not verbs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.1 Package Names

Names of packages that are to be made widely available should be formed as described in [§7.7](#). Such names are always qualified names whose first identifier consists of two or three uppercase letters that name an Internet domain, such as `COM`, `EDU`, `GOV`, `MIL`, `NET`, `ORG`, or a two-letter ISO country code such as `UK` or `JP`. Here are examples of hypothetical unique names that might be formed under this convention:

```
COM.JavaSoft.jag.Oak
ORG.NPR.pledge.driver
UK.ac.city.rugby.game
```

Names of packages intended only for local use should have a first identifier that begins with a lowercase letter, but that first identifier specifically should not be the identifier `java`; package names that start with the identifier `java` are reserved to JavaSoft for naming standard Java packages.

When package names occur in expressions:

- If a package name is hidden by a field declaration, then `import` declarations ([§7.5](#)) can usually be used to make available the type names declared in that package.
- If a package name is hidden by a declaration of a parameter or local variable, then the name of the parameter or local variable can be changed without affecting other Java code.
- The first component of a package name is normally not easily mistaken for a type name, as a type name normally begins with a single uppercase letter. (The Java language does not actually rely on case distinctions to determine whether a name is a package name or a type name. It is not possible for a type name to hide a package name.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.2 Class and Interface Type Names

Names of class types should be descriptive nouns or noun phrases, not overly long, in mixed case with the first letter of each word capitalized. For example:

```
ClassLoader  
SecurityManager  
Thread  
Dictionary  
BufferedInputStream
```

Likewise, names of interface types should be short and descriptive, not overly long, in mixed case with the first letter of each word capitalized. The name may be a descriptive noun or noun phrase, which is appropriate when an interface is used as if it were an abstract superclass, such as interfaces `java.io.DataInput` and `java.io.DataOutput`; or it may be an adjective describing a behavior, as for the interfaces `java.lang.Runnable` and `java.lang.Cloneable`.

Hiding involving class and interface type names is rare. Names of fields, parameters, and local variables normally do not hide type names because they conventionally begin with a lowercase letter whereas type names conventionally begin with an uppercase letter.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.3 Method Names

Method names should be verbs or verb phrases, in mixed case, with the first letter lowercase and the first letter of any subsequent words capitalized. Here are some additional specific conventions for method names:

- Methods to `get` and `set` an attribute that might be thought of as a variable *V* should be named `getV` and `setV`. An example is the methods `getPriority` (§20.20.22) and `setPriority` (§20.20.23) of class `java.lang.Thread`.
- A method that returns the length of something should be named `length`, as in class `java.lang.String` (§20.12.11).
- A method that tests a `boolean` condition *V* about an object should be named `isV`. An example is the method `isInterrupted` of class `java.lang.Thread` (§20.20.32).
- A method that converts its object to a particular format *F* should be named `toF`. Examples are the method `toString` of class `java.lang.Object` (§20.1.2) and the methods `toLocaleString` (§21.3.27) and `toGMTString` (§21.3.28) of class `java.util.Date`.

Whenever possible and appropriate, basing the names of methods in a new class on names in an existing class that is similar, especially a class from the standard Java Application Programming Interface classes, will make it easier to use.

Method names cannot hide or be hidden by other names (§6.5.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.4 Field Names

Names of fields that are not `final` should be in mixed case with a lowercase first letter and the first letters of subsequent words capitalized. Note that well-designed Java classes have very few `public` or `protected` fields, except for fields that are constants (`final static` fields) ([§6.8.5](#)).

Fields should have names that are nouns, noun phrases, or abbreviations for nouns. Examples of this convention are the fields `buf`, `pos`, and `count` of the class

`java.io.ByteArrayInputStream` ([§22.6](#)) and the field `bytesTransferred` of the class `java.io.InterruptedIOException` ([§22.30.1](#)).

Hiding involving field names is rare.

- If a field name hides a package name, then an `import` declaration ([§7.5](#)) can usually be used to make available the type names declared in that package.
- If a field name hides a type name, then a fully qualified name for the type can be used.
- Field names cannot hide method names.
- If a field name is hidden by a declaration of a parameter or local variable, then the name of the parameter or local variable can be changed without affecting other Java code.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.5 Constant Names

The names of constants in interface types should be, and `final` variables of class types may conventionally be, a sequence of one or more words, acronyms, or abbreviations, all uppercase, with components separated by underscore "_" characters. Constant names should be descriptive and not unnecessarily abbreviated. Conventionally they may be any appropriate part of speech. Examples of names for constants include `MIN_VALUE`, `MAX_VALUE`, `MIN_RADIX`, and `MAX_RADIX` of the class `java.lang.Character`.

A group of constants that represent alternative values of a set, or, less frequently, masking bits in an integer value, are sometimes usefully specified with a common acronym as a name prefix, as in:

```
interface ProcessStates {
    int PS_RUNNING = 0;
    int PS_SUSPENDED = 1;
}
```

Hiding involving constant names is rare:

- Constant names should be longer than three letters, so that they do not hide the initial component of a unique package name.
- Constant names normally have no lowercase letters, so they will not normally hide names of packages, types, or fields, whose names normally contain at least one lowercase letter.
- Constant names cannot hide method names, because they are distinguished syntactically.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


6.8.6 Local Variable and Parameter Names

Local variable and parameter names should be short, yet meaningful. They are often short sequences of lowercase letters that are not words. For example:

- Acronyms, that is the first letter of a series of words, as in `cp` for a variable holding a reference to a `ColoredPoint`
- Abbreviations, as in `buf` holding a pointer to a `buffer` of some kind
- Mnemonic terms, organized in some way to aid memory and understanding, typically by using a set of local variables with conventional names patterned after the names of parameters to widely used classes. For example:
 - `in` and `out`, whenever some kind of input and output are involved, patterned after the fields of `java.lang.System`
 - `off` and `len`, whenever an offset and length are involved, patterned after the parameters to the `read` and `write` methods of the interfaces `DataInput` and `DataOutput` of `java.io`

One-character local variable or parameter names should be avoided, except for temporary and looping variables, or where a variable holds an undistinguished value of a type. Conventional one-character names are:

- `b` for a `byte`
- `c` for a `char`
- `d` for a `double`
- `e` for an `Exception`
- `f` for a `float`
- `i`, `j`, and `k` for integers
- `l` for a `long`
- `o` for an `Object`
- `s` for a `String`
- `v` for an arbitrary value of some type

Local variable or parameter names that consist of only two or three uppercase letters should be avoided to avoid potential conflicts with the initial country codes and domain names that are the first component of unique package names ([§7.7](#)).

*What's in a name? That which we call a rose
By any other name would smell as sweet.*

--William Shakespeare, *Romeo and Juliet* (c. 1594), Act II, scene ii

Rose is a rose is a rose is a rose.

--Gertrude Stein, "Sacred Emily" (1913), in *Geographies and Plays*

. . . stat rosa pristina nomine, nomina nuda tenemus.

--Bernard of Morlay, *De contemptu mundi* (12th century),
quoted in Umberto Eco, *The Name of the Rose* (1980)

*Rose, Rose, bo-Bose,
Banana-fana fo-Fose,
Fee, fi, mo-Mose--
--Rose!*

--Lincoln Chase and Shirley Ellison, *The Name Game*
(#3 pop single in the U.S., January, 1965), as applied to the name "Rose"

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Packages

Good things come in small packages.

--Traditional proverb

Java programs are organized as sets of packages. Each package has its own set of names for types, which helps to prevent name conflicts. A type is accessible (§6.6) outside the package that declares it only if the type is declared `public`.

The naming structure for packages is hierarchical (§7.1). The members of a package are class and interface types (§7.6), which are declared in compilation units of the package, and subpackages, which may contain compilation units and subpackages of their own.

A package can be stored in a file system (§7.2.1) or in a database (§7.2.2). Packages that are stored in a file system have certain constraints on the organization of their compilation units to allow a simple implementation to find classes easily. In either case, the set of packages available to a Java program is determined by the host system, but must always include at least the three standard packages `java.lang`, `java.util`, and `java.io` as specified in Chapters 20, 21, and 22. In most host environments, the standard packages `java.applet`, `java.awt`, and `java.net`, which are not described in this specification, are also available to Java programs.

A package consists of a number of compilation units (§7.3). A compilation unit automatically has access to all types declared in its package and also automatically imports each of the types declared in the predefined package `java.lang`.

A compilation unit has three parts, each of which is optional:

- A `package` declaration (§7.4), giving the fully qualified name (§6.7) of the package to which the compilation unit belongs
- `import` declarations (§7.5) that allow types from other packages to be referred to using their simple names
- Type declarations (§7.6) of class and interface types

For small programs and casual development, a package can be unnamed (§7.4.2) or have a simple name, but if Java code is to be widely distributed, unique package names should be chosen (§7.7). This can prevent the conflicts that would otherwise occur if two development groups happened to pick the same package name and these packages were later to be used in a single program.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.1 Package Members

A *package* can have members of either or both of the following kinds:

- Subpackages of the package
- Types declared in the compilation units (§7.3) of the package

For example, in the standard Java Application Programming Interface:

- The package `java` has subpackages `awt`, `applet`, `io`, `lang`, `net`, and `util`, but no compilation units.
- The package `java.awt` has a subpackage named `image`, as well as a number of compilation units containing declarations of class and interface types.

If the fully qualified name (§6.7) of a package is *P*, and *Q* is a subpackage of *P*, then *P.Q* is the fully qualified name of the subpackage.

The subpackages of package `java` named `lang`, `util`, and `io` (whose fully qualified package names are therefore `java.lang`, `java.util`, and `java.io`) are a standard part of every Java implementation and are specified in Chapters 20, 21, and 22. Many Java implementations will include the entire set of `java` packages defined in the series of books *The Java Application Programming Interface*.

A package may not contain a type declaration and a subpackage of the same name, or a compile-time error results. Here are some examples:

- Because the package `java.awt` has a subpackage `image`, it cannot (and does not) contain a declaration of a class or interface type named `image`.
- If there is a package named `mouse` and a type `Button` in that package (which then might be referred to as `mouse.Button`), then there cannot be any package with the fully qualified name `mouse.Button` or `mouse.Button.Click`.
- If `COM.Sun.java.jag` is the fully qualified name of a type, then there cannot be any package whose fully qualified name is either `COM.Sun.java.jag` or `COM.Sun.java.jag.scrabble`.

The hierarchical naming structure for packages is intended to be convenient for organizing related packages in a conventional manner, but has no significance in the Java language itself other than the prohibition against a package having a subpackage with the same simple name as a type declared in that package. There is no special access relationship in the Java language between a package named `oliver` and another package named `oliver.twist`, or between packages named `evelyn.wood` and `evelyn.Waugh`. For example, the code in a package named `oliver.twist` has no better access to the types declared within package `oliver` than code in any other package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.2 Host Support for Packages

Each Java host determines how packages, compilation units, and subpackages are created and stored; which top-level package names are in scope in a particular compilation; and which packages are accessible.

The packages may be stored in a local file system in simple implementations of Java. Other implementations may use a distributed file system or some form of database to store Java source and/or binary code.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.2.1 Storing Packages in a File System

As an extremely simple example, all the Java packages and source and binary code on a system might be stored in a single directory and its subdirectories. Each immediate subdirectory of this directory would represent a top-level package, that is, one whose fully qualified name consists of a single simple name. The directory might contain the following immediate subdirectories:

```
COM
gls
jag
java
wnj
```

where directory `java` would contain the standard Java Application Programming Interface packages that are part of every standard Java system; the directories `jag`, `gls`, and `wnj` might contain packages that the three authors of this specification created for their personal use and to share with each other within this small group; and the directory `COM` would contain packages procured from companies that used the conventions described in [§7.7](#) to generate unique names for their packages.

Continuing the example, the directory `java` would probably contain at least the following subdirectories:

```
applet
awt
io
lang
net
util
```

corresponding to the standard packages `java.applet`, `java.awt`, `java.io`, `java.lang`, `java.net`, and `java.util` that are defined as part of the standard Java Application Programming Interface.

Still continuing the example, if we were to look inside the directory `util`, we might see the following files:

```
BitSet.java
    Observable.java
BitSet.class
    Observable.class
Date.java
    Observer.java
Date.class
    Observer.class
Dictionary.java
    Properties.java
Dictionary.class
```



```
Properties.class
EmptyStackException.java
    Random.java
EmptyStackException.class
    Random.class
Enumeration.java
    Stack.java
Enumeration.class
    Stack.class
Hashtable.java
    StringTokenizer.java
Hashtable.class
    StringTokenizer.class
NoSuchElementException.java
    Vector.java
NoSuchElementException.class
    Vector.class
```

where each of the `.java` files contains the source for a compilation unit (§7.3) that contains the definition of a class or interface whose binary compiled form is contained in the corresponding `.class` file.

Under this simple organization of packages, an implementation of Java would transform a package name into a pathname by concatenating the components of the package name, placing a file name separator (directory indicator) between adjacent components. For example, if this simple organization were used on a UNIX system, where the file name separator is `/`, the package name:

```
jag.scrabble.board
```

would be transformed into the directory name:

```
jag/scrabble/board
```

and:

```
COM.Sun.sunsoft.DOE
```

would be transformed to the directory name:

```
COM/Sun/sunsoft/DOE
```

In fact, the standard JavaSoft Java Developer's Kit on UNIX differs from the very simple discipline described here only in that it provides a `CLASSPATH` environment variable that specifies a set of directories, each of which is treated like the single directory described here. These directories are searched in order for definitions of named packages and types.

A package name component or class name might contain a character that cannot correctly appear in a host file system's ordinary directory name, such as a Unicode character on a system that allows only ASCII characters in file names. As a convention, the character can be escaped by using, say, the `@` character followed by four hexadecimal digits giving the numeric value of the character, as in

the `\uxxxx` escape ([§3.3](#)), so that the package name:

```
children.activities.crafts.papierM\u00e2ch\u00e9
```

which can also be written using full Unicode as:

```
children.activities.crafts.papierMâché;
```

might be mapped to the directory name:

```
children/activities/crafts/papierM@00e2ch@00e9
```

If the `@` character is not a valid character in a file name for some given host file system, then some other character that is not valid in a Java identifier could be used instead.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.2.2 Storing Packages in a Database

A host system may store packages and their compilation units and subpackages in a database.

Java allows such a database to relax the restrictions (§7.6) on compilation units in file-based implementations. For example, a system that uses a database to store packages need not enforce a maximum of one `public` class or interface per compilation unit. Systems that use a database must, however, provide an option to convert a Java program to a form that obeys the restrictions, for purposes of export to file-based implementations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.3 Compilation Units

CompilationUnit is the goal symbol (§2.1) for the syntactic grammar (§2.3) of Java programs. It is defined by the following productions:

CompilationUnit:

*PackageDeclaration*opt *ImportDeclarations*opt *TypeDeclarations*opt

ImportDeclarations:

ImportDeclaration

ImportDeclarations *ImportDeclaration*

TypeDeclarations:

TypeDeclaration

TypeDeclarations *TypeDeclaration*

Types declared in different compilation units can depend on each other, circularly. A Java compiler must arrange to compile all such types at the same time.

A *compilation unit* consists of three parts, each of which is optional:

- A *package* declaration (§7.4), giving the fully qualified name (§6.7) of the package to which the compilation unit belongs
- *import* declarations (§7.5) that allow types from other packages to be referred to using their simple names
- *Type* declarations (§7.6) of class and interface types

Every compilation unit automatically and implicitly imports every `public` type name declared in the predefined package `java.lang`, so that the names of all those types are available as simple names, as described in §7.5.3.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

7.4 Package Declarations

A package declaration appears within a compilation unit to indicate the package to which the compilation unit belongs. A compilation unit that has no package declaration is part of an unnamed package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.4.1 Named Packages

A *package declaration* in a compilation unit specifies the name (\$6.2) of the package to which the compilation unit belongs.

PackageDeclaration:

```
package PackageName ;
```

The package name mentioned in a package declaration must be the fully qualified name (\$6.7) of the package.

If a type named *T* is declared in a compilation unit of a package whose fully qualified name is *P*, then the fully qualified name of the type is *P.T*; thus in the example:

```
package wnj.points;  
class Point { int x, y; }
```

the fully qualified name of class `Point` is `wnj.points.Point`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.4.2 Unnamed Packages

A compilation unit that has no package declaration is part of an unnamed package. As an example, the compilation unit:

```
class FirstCall {
    public static void main(String[] args) {
        System.out.println("Mr. Watson, come here. "
                           + "I want you.");
    }
}
```

defines a very simple compilation unit as part of an unnamed package.

A Java system must support at least one unnamed package; it may support more than one unnamed package but is not required to do so. Which compilation units are in each unnamed package is determined by the host system.

In Java systems that use a hierarchical file system for storing packages, one typical strategy is to associate an unnamed package with each directory; only one unnamed package is available at a time, namely the one that is associated with the "current working directory." The precise meaning of "current working directory" depends on the host system.

Unnamed packages are provided by Java principally for convenience when developing small or temporary applications or when just beginning development.

Caution must be taken when using unnamed packages. It is possible for a compilation unit in a named package to import a type from an unnamed package, but the compiled version of this compilation unit will likely then work only when that particular unnamed package is "current." For this reason, it is strongly recommended that compilation units of named packages never import types from unnamed packages. It is also recommended that any type declared in an unnamed package not be declared `public`, to keep them from accidentally being imported by a named package.

It is recommended that a Java system provide safeguards against unintended consequences in situations where compilation units of named packages import types from unnamed packages. One strategy is to provide a way to associate with each named package at most one unnamed package, and then to detect and warn about situations in which a named package is used by more than one unnamed package. It is specifically not required-indeed, it is strongly discouraged-for an implementation to support use of a named package by more than one unnamed package by maintaining multiple compiled versions of the named package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.4.3 Scope and Hiding of a Package Name

Which top-level package names are in scope (\$6.3, \$6.5) is determined by conventions of the host system.

Package names never hide other names.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

7.4.4 Access to Members of a Package

Whether access to members of a package is allowed is determined by the host system. The package `java` should always be accessible, and its standard subpackages `lang`, `io`, and `util` should always be accessible.

It is strongly recommended that the protections of a file system or database used to store Java programs be set to make all compilation units of a package available whenever any of the compilation units is available.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.5 Import Declarations

An *import declaration* allows a type declared in another package to be referred to by a simple name ([§6.2](#)) that consists of a single identifier. Without the use of an appropriate `import` declaration, the only way to refer to a type declared in another package is to use its fully qualified name ([§6.7](#)).

ImportDeclaration:

SingleTypeImportDeclaration

TypeImportOnDemandDeclaration

A single-type-import declaration ([§7.5.1](#)) imports a single type, by mentioning its fully qualified name. A type-import-on-demand declaration ([§7.5.2](#)) imports all the `public` types of a named package as needed.

An `import` declaration makes types available by their simple names only within the compilation unit that actually contains the `import` declaration. The scope of the name(s) it introduces specifically does not include the `package` statement, other `import` statements in the current compilation unit, or other compilation units in the same package. Please see [§7.5.4](#) for an illustrative example.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.5.1 Single-Type-Import Declaration

A *single-type-import declaration* imports a single type by giving its fully qualified name, making it available under a simple name in the class and interface declarations of its compilation unit.

SingleTypeImportDeclaration:

```
import TypeName ;
```

The *TypeName* must be the fully qualified name of a class or interface type; a compile-time error occurs if the named type does not exist. If the named type is not in the current package, then it must be accessible ([§6.6](#))-in an accessible package and declared `public` ([§8.1.2](#), [§9.1.2](#))-or a compile-time error occurs.

The example:

```
import java.util.Vector;
```

causes the simple name `Vector` to be available within the class and interface declarations in a compilation unit. Thus, the simple name `Vector` refers to the type `Vector` in the package `java.util` in all places where it is not hidden ([§6.3](#)) by a declaration of a field, parameter, or local variable with the same name.

If two single-type-import declarations in the same compilation unit attempt to import types with the same simple name, then a compile-time error occurs, unless the two types are the same type, in which case the duplicate declaration is ignored. If another type with the same name is otherwise declared in the current compilation unit except by a type-import-on-demand declaration ([§7.5.2](#)), then a compile-time error occurs.

So the sample program:

```
import java.util.Vector;

class Vector { Object[] vec; }
```

causes a compile-time error because of the duplicate declaration of `Vector`, as does:

```
import java.util.Vector;

import myVector.Vector;
```

where `myVector` is a package containing the compilation unit:

```
package myVector;
```



```
public class Vector { Object[] vec; }
```

The compiler keeps track of types by their fully qualified names (§6.7). Simple names and fully qualified names may be used interchangeably whenever they are both available.

Note that an import statement cannot import a subpackage, only a type. For example, it does not work to try to import `java.util` and then use the name `util.Random` to refer to the type `java.util.Random`:

```
import java.util;  
    // incorrect: compile-time error  
  
class Test { util.Random generator; }
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.5.2 Type-Import-on-Demand Declaration

A *type-import-on-demand declaration* allows all `public` types declared in the package named by a fully qualified name to be imported as needed.

TypeImportOnDemandDeclaration:

```
import PackageName . * ;
```

It is a compile-time error for a type-import-on-demand declaration to name a package that is not accessible (§6.6), as determined by the host system (§7.2). Two or more type-import-on-demand declarations in the same compilation unit may name the same package; the effect is as if there were exactly one such declaration. It is not a compile-time error to name the current package or `java.lang` in a type-import-on-demand declaration, even though they are already imported; the duplicate type-import-on-demand declaration is ignored.

The example:

```
import java.util.*;
```

causes the simple names of all `public` types declared in the package `java.util` to be available within the class and interface declarations of the compilation unit. Thus, the simple name `Vector` refers to the type `Vector` in the package `java.util` in all places where it is not hidden (§6.3) by a single-type-import declaration of a type whose simple name is `Vector`; by a type named `Vector` and declared in the package to which the compilation unit belongs; or by a declaration of a field, parameter, or local variable named `Vector`. (It would be unusual for any of these conditions to occur.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.5.3 Automatic Imports

Each compilation unit automatically imports each of the `public` type names declared in the predefined package `java.lang`, as if the declaration:

```
import java.lang.*;
```

appeared at the beginning of each compilation unit, immediately following any `package` statement.

The full specification of `java.lang` is given in Chapter [20](#). The following `public` types are defined in `java.lang`:

AbstractMethodError	
LinkageError	
ArithmeticException	
Long	
ArrayStoreException	
Math	
Boolean	
NegativeArraySizeException	
Character	
NoClassDefFoundError	
Class	
NoSuchFieldError	
ClassCastException	
NoSuchMethodError	
ClassCircularityError	
NullPointerException	
ClassFormatError	
Number	
ClassLoader	
NumberFormatException	
ClassNotFoundException	
Object	
CloneNotSupportedException	
OutOfMemoryError	
Cloneable	Process
Compiler	Runnable
Double	Runtime
Error	
RuntimeException	
Exception	
SecurityException	
ExceptionInInitializerError	
SecurityManager	
Float	
StackOverflowError	
IllegalAccessError	
String	
IllegalAccessError	
StringBuffer	
IllegalArgumentException	

System
IllegalMonitorStateException
Thread
IllegalThreadStateException
ThreadDeath
IncompatibleClassChangeError
ThreadGroup
IndexOutOfBoundsException
Throwable
InstantiationError
UnknownError
InstantiationException
UnsatisfiedLinkError
Integer
VerifyError
InternalError
VirtualMachineError
InterruptedException

{ewl msdncd.dll, ewcright, /c"Microsoft"}

7.5.4 A Strange Example

Package names and type names are usually different under the naming conventions described in [§6.8](#). Nevertheless, in a contrived example where there is an unconventionally-named package `Vector`, which declares a public class named `Mosquito`:

```
package Vector;

public class Mosquito { int capacity; }
```

and then the compilation unit:

```
package strange.example;

import java.util.Vector;

import Vector.Mosquito;

class Test {
    public static void main(String[] args) {
        System.out.println(new Vector().getClass());
        System.out.println(new Mosquito().getClass());
    }
}
```

the single-type-import declaration ([§7.5.1](#)) importing class `Vector` from package `java.util` does not prevent the package name `Vector` from appearing and being correctly recognized in subsequent `import` declarations. The example compiles and produces the output:

```
class java.util.Vector
class Vector.Mosquito
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.6 Type Declarations

A type declaration declares a class type (§8) or an interface type (§9):

TypeDeclaration:

ClassDeclaration

InterfaceDeclaration

;

A Java compiler must ignore extra ";" tokens appearing at the level of type declarations. Stray semicolons are permitted in Java solely as a concession to C++ programmers who are used to writing:

```
class date { int month, day, year; };
```

(In C++, but not in Java, one can provide a comma-separated list of identifiers in order to declare variables between the "}" and the ";".) Extra semicolons should not be used in new Java code. Software that reformats Java code can delete them.

By default, the types declared in a package are accessible only within the compilation units of that package, but a type may be declared to be `public` to grant access to the type from code in other packages (§6.6, §8.1.2, §9.1.2).

A Java implementation must keep track of types within packages by their fully qualified names (§6.7). Multiple ways of naming a type must be expanded to fully qualified names to make sure that such names are understood as referring to the same type. For example, if a compilation unit contains the single-type-import declaration (§7.5.1):

```
import java.util.Vector;
```

then within that compilation unit the simple name `Vector` and the fully qualified name `java.util.Vector` refer to the same type.

When Java packages are stored in a file system (§7.2.1), the host system may choose to enforce the restriction that it is a compile-time error if a type is not found in a file under a name composed of the type name plus an extension (such as `.java` or `.jav`) if either of the following is true:

- The type is referred to by code in other compilation units of the package in which the type is declared.
- The type is declared `public` (and therefore is potentially accessible from code in other packages).

This restriction implies that there must be at most one such type per compilation unit. This restriction makes it easy for a Java compiler and Java Virtual Machine to find a named class within a package; for example, the source code for a `public` type `wet.sprocket.Toad` would be found in a file `Toad.java` in the directory `wet/sprocket`, and the corresponding object code would be found in

the file `Toad.class` in the same directory.

When Java packages are stored in a database (\$7.2.2), the host system need not enforce such restrictions.

In practice, many Java programmers choose to put each class or interface type in its own compilation unit, whether or not it is `public` or is referred to by code in other compilation units.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


7.7 Unique Package Names

*Did I ever tell you that Mrs. McCave
Had twenty-three sons and she named them all "Dave"?
Well, she did. And that wasn't a smart thing to do. . . .*

-Dr. Seuss (Theodore Geisel), *Too Many Daves* (1961)

Developers should take steps to avoid the possibility of two published packages having the same name by choosing *unique package names* for packages that are widely distributed. This allows packages to be easily and automatically installed and catalogued. This section specifies a standard convention, not enforced by a Java compiler, for generating such unique package names. Java systems are encouraged to provide automatic support for converting a set of packages from local and casual package names to the unique name format described here.

If unique package names are not used, then package name conflicts may arise far from the point of creation of either of the conflicting packages. This may create a situation that is difficult or impossible for the user or programmer to resolve. The class `ClassLoader` (§20.14) of the standard Java Virtual Machine environment can be used to isolate packages with the same name from each other in those cases where the packages will have constrained interactions, but not in a way that is transparent to a naive Java program.

You form a unique package name by first having (or belonging to an organization that has) an Internet domain name, such as `Sun.COM`. You then reverse this name, component by component, to obtain, in this example, `COM.Sun`, and use this as a prefix for your package names, using a convention developed within your organization to further administer package names.

Such a convention might specify that certain directory name components be division, department, project, machine, or login names. Some possible examples:

```
COM.Sun.sunsoft.DOE
COM.Sun.java.jag.scrabble
COM.Apple.quicktime.v2
EDU.cmu.cs.bovik.cheese
GOV.whitehouse.socks.mousefinder
```

The first component of a unique package name is always written in all-uppercase ASCII letters and should be one of the top-level domain names, currently `COM`, `EDU`, `GOV`, `MIL`, `NET`, `ORG`, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. For more information, refer to the documents stored at `ftp://rs.internic.net/rfc`, for example, `rfc920.txt` and `rfc1032.txt`.

The name of a package is not meant to imply anything about where the package is stored within the Internet; for example, a package named `EDU.cmu.cs.bovik.cheese` is not necessarily obtainable from Internet address `cmu.EDU` or from `cs.cmu.EDU` or from `bovik.cs.cmu.EDU`. The Java convention for generating unique package names is merely a way to piggyback a package naming convention on top of an existing, widely known unique name registry instead of having to create a separate registry for Java package names.

If you need to get a new Internet domain name, you can get an application form from `ftp://ftp.internic.net` and submit the complete forms by E-mail to `domreg@internic.net`. To find out what the currently registered domain names are, you can `telnet` to `rs.internic.net` and use the `whois` facility.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Classes

class 1. The noun class derives from < Medieval French and French classe from Latin classis, probably originally a summons, hence a summoned collection of persons, a group liable to be summoned: perhaps for callassis from calare, to call, hence to summon.

--Eric Partridge, Origins: A Short Etymological Dictionary of Modern English

Class declarations define new reference types and describe how they are implemented (§8.1).

The name of a class has as its scope all type declarations in the package in which the class is declared (§8.1.1). A class may be declared `abstract` (§8.1.2.1) and must be declared `abstract` if it is incompletely implemented; such a class cannot be instantiated, but can be extended by subclasses. A class may be declared `final` (§8.1.2.2), in which case it cannot have subclasses. If a class is declared `public`, then it can be referred to from other packages.

Each class except `Object` is an extension of (that is, a subclass of) a single existing class (§8.1.3) and may implement interfaces (§8.1.4).

The body of a class declares members (fields and methods), static initializers, and constructors (§8.1.5). The scope of the name of a member is the entire declaration of the class to which the member belongs. Field, method, and constructor declarations may include the access modifiers (§6.6) `public`, `protected`, or `private`. The members of a class include both declared and inherited members (§8.2). Newly declared fields can hide fields declared in a superclass or superinterface. Newly declared methods can hide, implement, or override methods declared in a superclass or superinterface.

Field declarations (§8.3) describe class variables, which are incarnated once, and instance variables, which are freshly incarnated for each instance of the class. A field may be declared `final` (§8.3.1.2), in which case it cannot be assigned to except as part of its declaration. Any field declaration may include an initializer; the declaration of a `final` field must include an initializer.

Method declarations (§8.4) describe code that may be invoked by method invocation expressions (§15.11). A class method is invoked relative to the class type; an instance method is invoked with respect to some particular object that is an instance of the class type. A method whose declaration does not indicate how it is implemented must be declared `abstract`. A method may be declared `final` (§8.4.3.3), in which case it cannot be hidden or overridden. A method may be implemented by platform-dependent `native` code (§8.4.3.4). A `synchronized` method (§8.4.3.5) automatically locks an object before executing its body and automatically unlocks the object on return, as if by use of a `synchronized` statement (§14.17), thus allowing its activities to be synchronized with those of other threads (§17).

Method names may be overloaded (§8.4.7).

Static initializers (§8.5) are blocks of executable code that may be used to help initialize a class when it is first loaded (§12.4).

Constructors (§8.6) are similar to methods, but cannot be invoked directly by a method call; they are used to initialize new class instances. Like methods, they may be overloaded (§8.6.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1 Class Declaration

A *class declaration* specifies a new reference type:

ClassDeclaration:

*ClassModifiers*opt *class* *Identifier* *Super*opt *Interfaces*opt *ClassBody*

If a class is declared in a named package ([§7.4.1](#)) with fully qualified name *P* ([§6.7](#)), then the class has the fully qualified name *P.Identifier*. If the class is in an unnamed package ([§7.4.2](#)), then the class has the fully qualified name *Identifier*. In the example:

```
class Point { int x, y; }
```

the class `Point` is declared in a compilation unit with no `package` statement, and thus `Point` is its fully qualified name, whereas in the example:

```
package vista;  
class Point { int x, y; }
```

the fully qualified name of the class `Point` is `vista.Point`. (The package name `vista` is suitable for local or personal use; if the package were intended to be widely distributed, it would be better to give it a unique package name ([§7.7](#)).)

A compile-time error occurs if the *Identifier* naming a class appears as the name of any other class type or interface type declared in the same package ([§7.6](#)).

A compile-time error occurs if the *Identifier* naming a class is also declared as a type by a single-type-import declaration ([§7.5.1](#)) in the compilation unit ([§7.3](#)) containing the class declaration.

In the example:

```
package test;  
  
import java.util.Vector;  
  
class Point {  
    int x, y;  
}  
  
interface Point {  
    // compile-time error #1  
    int getR();  
    int getTheta();  
}
```



```

}

class Vector { Point[] pts; }
// compile-time error #2

```

the first compile-time error is caused by the duplicate declaration of the name `Point` as both a class and an interface in the same package. A second error detected at compile time is the attempt to declare the name `Vector` both by a class type declaration and by a single-type-import declaration.

Note, however, that it is not an error for the *Identifier* that names a class also to name a type that otherwise might be imported by a type-import-on-demand declaration ([§7.5.2](#)) in the compilation unit ([§7.3](#)) containing the class declaration. In the example:

```

package test;

import java.util.*;

class Vector { Point[] pts; }
// not a compile-time error

```

the declaration of the class `Vector` is permitted even though there is also a class `java.util.Vector`. Within this compilation unit, the simple name `Vector` refers to the class `test.Vector`, not to `java.util.Vector` (which can still be referred to by code within the compilation unit, but only by its fully qualified name).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1.1 Scope of a Class Type Name

The *Identifier* in a class declaration specifies the name of the class. This class name has as its scope (§6.3) the entire package in which the class is declared. As an example, the compilation unit:

```
package points;
```

```
class Point {
    int x, y; //
coordinates
    PointColor color; //
color of this point
    Point next; // next
point with this color

    static int nPoints;
}
```

```
class PointColor {
    Point first; //
first point with this color
    PointColor(int color) {
        this.color = color;
    }
    private int color;
    // color components
}
```

defines two classes that use each other in the declarations of their class members. Because the class type names `Point` and `PointColor` have the entire package `points`, including the entire current compilation unit, as their scope, this example compiles correctly—that is, forward reference is not a problem.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1.2 Class Modifiers

A class declaration may include *class modifiers*.

ClassModifiers:

ClassModifier

ClassModifiers ClassModifier

ClassModifier: one of

public abstract final

The access modifier `public` is discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in a class declaration. If two or more class modifiers appear in a class declaration, then it is customary, though not required, that they appear in the order consistent with that shown above in the production for *ClassModifier*.

8.1.2.1 abstract Classes

An `abstract` class is a class that is incomplete, or to be considered incomplete. Only `abstract` classes may have `abstract` methods ([§8.4.3.1](#), [§9.4](#)), that is, methods that are declared but not yet implemented. If a class that is not `abstract` contains an `abstract` method, then a compile-time error occurs. A class has `abstract` methods if any of the following is true:

- It explicitly contains a declaration of an `abstract` method ([§8.4.3](#)).
- It inherits an `abstract` method from its direct superclass ([§8.1.3](#)).
- A direct superinterface ([§8.1.4](#)) of the class declares or inherits a method (which is therefore necessarily `abstract`) and the class neither declares nor inherits a method that implements it.

In the example:

```
abstract class Point {
    int x = 1, y = 1;
    void move(int dx, int dy) {
        x += dx;
        y += dy;
        alert();
    }
    abstract void alert();
}

abstract class ColoredPoint extends Point {
    int color;
}
```



```
class SimplePoint extends Point {
    void alert() { }
}
```

a class `Point` is declared that must be declared `abstract`, because it contains a declaration of an abstract method named `alert`. The subclass of `Point` named `ColoredPoint` inherits the abstract method `alert`, so it must also be declared `abstract`. On the other hand, the subclass of `Point` named `SimplePoint` provides an implementation of `alert`, so it need not be `abstract`.

A compile-time error occurs if an attempt is made to create an instance of an `abstract` class using a class instance creation expression (§15.8). An attempt to instantiate an `abstract` class using the `newInstance` method of class `Class` (§20.3.6) will cause an `InstantiationException` (§11.5.1) to be thrown. Thus, continuing the example just shown, the statement:

```
Point p = new Point();
```

would result in a compile-time error; the class `Point` cannot be instantiated because it is `abstract`. However, a `Point` variable could correctly be initialized with a reference to any subclass of `Point`, and the class `SimplePoint` is not `abstract`, so the statement:

```
Point p = new SimplePoint();
```

would be correct.

A subclass of an `abstract` class that is not itself `abstract` may be instantiated, resulting in the execution of a constructor for the `abstract` class and, therefore, the execution of the field initializers for instance variables of that class. Thus, in the example just given, instantiation of a `SimplePoint` causes the default constructor and field initializers for `x` and `y` of `Point` to be executed.

It is a compile-time error to declare an `abstract` class type such that it is not possible to create a subclass that implements all of its `abstract` methods. This situation can occur if the class would have as members two `abstract` methods that have the same method signature (§8.4.2) but different return types. As an example, the declarations:

```
interface Colorable { void setColor(int color); }

abstract class Colored implements Colorable {
    abstract int setColor(int color);
}
```

result in a compile-time error: it would be impossible for any subclass of class `Colored` to provide an implementation of a method named `setColor`, taking one argument of type `int`, that can satisfy both `abstract` method specifications, because the one in interface `Colorable` requires the same method to return no value, while the one in class `Colored` requires the same method to return a value of type `int` (§8.4).

A class type should be declared `abstract` only if the intent is that subclasses can be created to

complete the implementation. If the intent is simply to prevent instantiation of a class, the proper way to express this is to declare a constructor (§8.6.8) of no arguments, make it `private`, never invoke it, and declare no other constructors. A class of this form usually contains class methods and variables. The class `java.lang.Math` is an example of a class that cannot be instantiated; its declaration looks like this:

```
public final class Math {  
  
    private Math() { }                                // never  
    instantiate this class  
  
    declarations of class variables and methods  
  
}
```

8.1.2.2 final Classes

A class can be declared `final` if its definition is complete and no subclasses are desired or required. A compile-time error occurs if the name of a `final` class appears in the `extends` clause (§8.1.3) of another `class` declaration; this implies that a `final` class cannot have any subclasses. A compile-time error occurs if a class is declared both `final` and `abstract`, because the implementation of such a class could never be completed (§8.1.2.1).

Because a `final` class never has any subclasses, the methods of a `final` class are never overridden (§8.4.6.1).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1.3 Superclasses and Subclasses

The optional `extends` clause in a class declaration specifies the *direct superclass* of the current class. A class is said to be a *direct subclass* of the class it extends. The direct superclass is the class from whose implementation the implementation of the current class is derived. The `extends` clause must not appear in the definition of the class `java.lang.Object` (§20.1), because it is the primordial class and has no direct superclass. If the class declaration for any other class has no `extends` clause, then the class has the class `java.lang.Object` as its implicit direct superclass.

Super:

`extends ClassType`

The following is repeated from §4.3 to make the presentation here clearer:

ClassType:

`TypeName`

The *ClassType* must name an accessible (§6.6) class type, or a compile-time error occurs. All classes in the current package are accessible. Classes in other packages are accessible if the host system permits access to the package (§7.2) and the class is declared `public`. If the specified *ClassType* names a class that is `final` (§8.1.2.2), then a compile-time error occurs; `final` classes are not allowed to have subclasses.

In the example:

```
class Point { int x, y; }
```

```
final class ColoredPoint extends Point { int color; }
```

```
class Colored3DPoint extends ColoredPoint { int z; } // error
```

the relationships are as follows:

- The class `Point` is a direct subclass of `java.lang.Object`.
- The class `java.lang.Object` is the direct superclass of the class `Point`.
- The class `ColoredPoint` is a direct subclass of class `Point`.
- The class `Point` is the direct superclass of class `ColoredPoint`.

The declaration of class `Colored3dPoint` causes a compile-time error because it attempts to extend the `final` class `ColoredPoint`.

The *subclass* relationship is the transitive closure of the direct subclass relationship. A class *A* is a subclass of class *C* if either of the following is true:

- *A* is the direct subclass of *C*.
- There exists a class *B* such that *A* is a subclass of *B*, and *B* is a subclass of *C*, applying this definition recursively.

Class *C* is said to be a *superclass* of class *A* whenever *A* is a subclass of *C*.

In the example:

```
class Point { int x, y; }
```

```
class ColoredPoint extends Point { int color; }
```

```
final class Colored3dPoint extends ColoredPoint { int z; }
```

the relationships are as follows:

- The class `Point` is a superclass of class `ColoredPoint`.
- The class `Point` is a superclass of class `Colored3dPoint`.
- The class `ColoredPoint` is a subclass of class `Point`.
- The class `ColoredPoint` is a superclass of class `Colored3dPoint`.
- The class `Colored3dPoint` is a subclass of class `ColoredPoint`.
- The class `Colored3dPoint` is a subclass of class `Point`.

A compile-time error occurs if a class is declared to be a subclass of itself. For example:

```
class Point extends ColoredPoint { int x, y; }
```

```
class ColoredPoint extends Point { int color; }
```

causes a compile-time error. If circularly declared classes are detected at run time, as classes are loaded ([§12.2](#)), then a `ClassCircularityError` is thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1.4 Superinterfaces

The optional `implements` clause in a class declaration lists the names of interfaces that are *direct superinterfaces* of the class being declared:

Interfaces:

implements InterfaceTypeList

InterfaceTypeList:

InterfaceType

InterfaceTypeList , InterfaceType

The following is repeated from [§4.3](#) to make the presentation here clearer:

InterfaceType:

TypeName

Each *InterfaceType* must name an accessible ([§6.6](#)) interface type, or a compile-time error occurs. All interfaces in the current package are accessible. Interfaces in other packages are accessible if the host system permits access to the package ([§7.4.4](#)) and the interface is declared `public`.

A compile-time error occurs if the same interface is mentioned two or more times in a single `implements` clause, even if the interface is named in different ways; for example, the code:

```
class Redundant implements java.lang.Cloneable, Cloneable {  
    int x;  
}
```

results in a compile-time error because the names `java.lang.Cloneable` and `Cloneable` refer to the same interface.

An interface type *I* is a *superinterface* of class type *C* if any of the following is true:

- *I* is a direct superinterface of *C*.
- *C* has some direct superinterface *J* for which *I* is a superinterface, using the definition of "superinterface of an interface" given in [§9.1.3](#).
- *I* is a superinterface of the direct superclass of *C*, using this definition recursively.

A class is said to *implement* all its superinterfaces.

In the example:


```

public interface Colorable {
    void setColor(int color);
    int getColor();
}

public interface Paintable extends Colorable {
    int MATTE = 0, GLOSSY = 1;
    void setFinish(int finish);
    int getFinish();
}

class Point { int x, y; }

class ColoredPoint extends Point implements Colorable {
    int color;
    public void setColor(int color) { this.color = color; }
    public int getColor() { return color; }
}

class PaintedPoint extends ColoredPoint implements Paintable
{
    int finish;
    public void setFinish(int finish) {
        this.finish = finish;
    }
    public int getFinish() { return finish; }
}

```

the relationships are as follows:

- The interface `Paintable` is a superinterface of class `PaintedPoint`.
- The interface `Colorable` is a superinterface of class `ColoredPoint` and of class `PaintedPoint`.
- The interface `Paintable` is a subinterface of the interface `Colorable`, and `Colorable` is a superinterface of `Paintable`, as defined in §9.1.3.

A class can have a superinterface in more than one way. In this example, the class `PaintedPoint` has `Colorable` as a superinterface both because it is a superinterface of `ColoredPoint` and because it is a superinterface of `Paintable`.

Unless the class being declared is `abstract`, the declarations of the methods defined in each direct superinterface must be implemented either by a declaration in this class or by an existing method declaration inherited from the direct superclass, because a class that is not `abstract` is not permitted to have abstract methods (§8.1.2.1).

Thus, the example:

```

interface Colorable {
    void setColor(int color);
}

```



```
        int getColor();  
    }
```

```
class Point { int x, y; };
```

```
class ColoredPoint extends Point implements Colorable {  
    int color;  
}
```

causes a compile-time error, because `ColoredPoint` is not an abstract class but it fails to provide an implementation of methods `setColor` and `getColor` of the interface `Colorable`.

It is permitted for a single method declaration in a class to implement methods of more than one superinterface. For example, in the code:

```
interface Fish { int getNumberOfScales(); }
```

```
interface Piano { int getNumberOfScales(); }
```

```
class Tuna implements Fish, Piano {  
    // You can tune a piano, but can you tuna fish?  
    int getNumberOfScales() { return 91; }  
}
```

the method `getNumberOfScales` in class `Tuna` has a name, signature, and return type that matches the method declared in interface `Fish` and also matches the method declared in interface `Piano`; it is considered to implement both.

On the other hand, in a situation such as this:

```
interface Fish { int getNumberOfScales(); }
```

```
interface StringBass { double getNumberOfScales(); }
```

```
class Bass implements Fish, StringBass {  
    // This declaration cannot be correct, no matter what type is used.  
    public ??? getNumberOfScales() { return 91; }  
}
```


it is impossible to declare a method named `getNumberOfScales` with the same signature and return type as those of both the methods declared in interface `Fish` and in interface `StringBass`, because a class can have only one method with a given signature (§8.4). Therefore, it is impossible for a single class to implement both interface `Fish` and interface `StringBass` (§8.4.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.1.5 Class Body and Member Declarations

A *class body* may contain declarations of members of the class, that is, fields (§8.3) and methods (§8.4). A class body may also contain static initializers (§8.5) and declarations of constructors (§8.6) for the class.

ClassBody:

{ ClassBodyDeclarationsopt }

ClassBodyDeclarations:

ClassBodyDeclaration

ClassBodyDeclarations ClassBodyDeclaration

ClassBodyDeclaration:

ClassMemberDeclaration

StaticInitializer

ConstructorDeclaration

ClassMemberDeclaration:

FieldDeclaration

MethodDeclaration

The scope of the name of a member declared in or inherited by a class type is the entire body of the class type declaration.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.2 Class Members

The members of a class type are all of the following:

- Members inherited from its direct superclass (§8.1.3), except in class `Object`, which has no direct superclass
- Members inherited from any direct superinterfaces (§8.1.4)
- Members declared in the body of the class (§8.1.5)

Members of a class that are declared `private` are not inherited by subclasses of that class. Only members of a class that are declared `protected` or `public` are inherited by subclasses declared in a package other than the one in which the class is declared.

Constructors and static initializers are not members and therefore are not inherited.

The example:

```
class Point {
    int x, y;
    private Point() { reset(); }
    Point(int x, int y) { this.x = x; this.y = y; }
    private void reset() { this.x = 0; this.y = 0; }
}

class ColoredPoint extends Point {
    int color;
    void clear() { reset(); }
                                     // error
}

class Test {
    public static void main(String[] args) {
        ColoredPoint c = new ColoredPoint(0, 0);
                                     // error
        c.reset();
                                     // error
    }
}
```

causes four compile-time errors:

- An error occurs because `ColoredPoint` has no constructor declared with two integer parameters, as requested by the use in `main`. This illustrates the fact that `ColoredPoint` does not inherit the constructors of its superclass `Point`.
- Another error occurs because `ColoredPoint` declares no constructors, and therefore a default constructor for it is automatically created (§8.6.7), and this default constructor is equivalent to:
`ColoredPoint() { super(); }`

which invokes the constructor, with no arguments, for the direct superclass of the class `ColoredPoint`. The error is that the constructor for `Point` that takes no arguments is `private`,

and therefore is not accessible outside the class `Point`, even through a superclass constructor invocation (§8.6.5).

- Two more errors occur because the method `reset` of class `Point` is `private`, and therefore is not inherited by class `ColoredPoint`. The method invocations in method `clear` of class `ColoredPoint` and in method `main` of class `Test` are therefore not correct.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.2.1 Examples of Inheritance

This section illustrates inheritance of class members through several examples.

8.2.1.1 Example: Inheritance with Default Access

Consider the example where the `points` package declares two compilation units:

```
package points;
```

```
public class Point {
    int x, y;

    public void move(int dx, int dy) { x += dx; y += dy; }
}
```

and:

```
package points;
```

```
public class Point3d extends Point {
    int z;
    public void move(int dx, int dy, int dz) {
        x += dx; y += dy; z += dz;
    }
}
```

and a third compilation unit, in another package, is:

```
import points.Point3d;
```

```
class Point4d extends Point3d {
    int w;
    public void move(int dx, int dy, int dz, int dw) {
        x += dx; y += dy; z += dz; w += dw; // compile-time errors
    }
}
```



```
    }  
}
```

Here both classes in the `points` package compile. The class `Point3d` inherits the fields `x` and `y` of class `Point`, because it is in the same package as `Point`. The class `Point4d`, which is in a different package, does not inherit the fields `x` and `y` of class `Point` or the field `z` of class `Point3d`, and so fails to compile.

A better way to write the third compilation unit would be:

```
import points.Point3d;
```

```
class Point4d extends Point3d {  
    int w;  
    public void move(int dx, int dy, int dz, int dw) {  
        super.move(dx, dy, dz); w += dw;  
    }  
}
```

using the `move` method of the superclass `Point3d` to process `dx`, `dy`, and `dz`. If `Point4d` is written in this way it will compile without errors.

8.2.1.2 Inheritance with public and protected

Given the class `Point`:

```
package points;
```

```
public class Point {  
    public int x, y;  
  
    protected int useCount = 0;  
  
    static protected int totalUseCount = 0;  
  
    public void move(int dx, int dy) {  
        x += dx; y += dy; useCount++; totalUseCount++;  
    }  
}
```



```
}
```

the public and protected fields `x`, `y`, `useCount` and `totalUseCount` are inherited in all subclasses of `Point`. Therefore, this test program, in another package, can be compiled successfully:

```
class Test extends points.Point {  
    public void moveBack(int dx, int dy) {  
        x -= dx; y -= dy; useCount++; totalUseCount++;  
    }  
}
```

8.2.1.3 Inheritance with private

In the example:

```
class Point {  
  
    int x, y;  
  
    void move(int dx, int dy) {  
        x += dx; y += dy; totalMoves++;  
    }  
  
    private static int totalMoves;  
  
    void printMoves() { System.out.println(totalMoves); }  
}  
  
class Point3d extends Point {  
  
    int z;  
  
    void move(int dx, int dy, int dz) {  
        super.move(dx, dy); z += dz; totalMoves++;  
    }  
}
```

the class variable `totalMoves` can be used only within the class `Point`; it is not inherited by the subclass `Point3d`. A compile-time error occurs at the point where method `move` of class `Point3d` tries to increment `totalMoves`.

8.2.1.4 Accessing Members of Inaccessible Classes

Even though a class might not be declared `public`, instances of the class might be available at run time to code outside the package in which it is declared if it has a `public` superclass or superinterface. An instance of the class can be assigned to a variable of such a `public` type. An invocation of a `public` method of the object referred to by such a variable may invoke a method of the class if it implements or overrides a method of the `public` superclass or superinterface. (In this situation, the method is necessarily declared `public`, even though it is declared in a class that is not `public`.)

Consider the compilation unit:

```
package points;
```

```
public class Point {
    public int x, y;
    public void move(int dx, int dy) {
        x += dx; y += dy;
    }
}
```

and another compilation unit of another package:

```
package morePoints;
```

```
class Point3d extends points.Point {
    public int z;
    public void move(int dx, int dy, int dz) {
        super.move(dx, dy); z += dz;
    }
}
```

```
public class OnePoint {
    static points.Point getOne() { return new Point3d(); }
}
```

An invocation `morePoints.OnePoint.getOne()` in yet a third package would return a `Point3d` that can be used as a `Point`, even though the type `Point3d` is not available outside the package `morePoints`. The method `move` could then be invoked for that object, which is permissible because method `move` of `Point3d` is `public` (as it must be, for any method that overrides a `public` method

must itself be `public`, precisely so that situations such as this will work out correctly). The fields `x` and `y` of that object could also be accessed from such a third package.

While the field `z` of class `Point3d` is `public`, it is not possible to access this field from code outside the package `morePoints`, given only a reference to an instance of class `Point3d` in a variable `p` of type `Point`. This is because the expression `p.z` is not correct, as `p` has type `Point` and class `Point` has no field named `z`; also, the expression `((Point3d)p).z` is not correct, because the class type `Point3d` cannot be referred to outside package `morePoints`. The declaration of the field `z` as `public` is not useless, however. If there were to be, in package `morePoints`, a `public` subclass `Point4d` of the class `Point3d`:

```
package morePoints;
```

```
public class Point4d extends Point3d {
    public int w;
    public void move(int dx, int dy, int dz, int dw) {
        super.move(dx, dy, dz); w += dw;
    }
}
```

then class `Point4d` would inherit the field `z`, which, being `public`, could then be accessed by code in packages other than `morePoints`, through variables and expressions of the `public` type `Point4d`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.3 Field Declarations

*Poetic fields encompass me around,
And still I seem to tread on classic ground.*
--Joseph Addison (1672-1719), *A Letter from Italy*

The variables of a class type are introduced by *field declarations*:

FieldDeclaration:

FieldModifiersopt Type VariableDeclarators ;

VariableDeclarators:

VariableDeclarator

VariableDeclarators , VariableDeclarator

VariableDeclarator:

VariableDeclaratorId

VariableDeclaratorId = VariableInitializer

VariableDeclaratorId:

Identifier

VariableDeclaratorId []

VariableInitializer:

Expression

ArrayInitializer

The *FieldModifiers* are described in §8.3.1. The *Identifier* in a *FieldDeclarator* may be used in a name to refer to the field. The name of a field has as its scope (§6.3) the entire body of the class declaration in which it is declared. More than one field may be declared in a single field declaration by using more than one declarator; the *FieldModifiers* and *Type* apply to all the declarators in the declaration. Variable declarations involving array types are discussed in §10.2.

It is a compile-time error for the body of a class declaration to contain declarations of two fields with the same name. Methods and fields may have the same name, since they are used in different contexts and are disambiguated by the different lookup procedures (§6.5).

If the class declares a field with a certain name, then the declaration of that field is said to *hide* (§6.3.1) any and all accessible declarations of fields with the same name in the superclasses and superinterfaces of the class.

If a field declaration hides the declaration of another field, the two fields need not have the same type.

A class inherits from its direct superclass and direct superinterfaces all the fields of the superclass and superinterfaces that are both accessible to code in the class and not hidden by a declaration in the class.

It is possible for a class to inherit more than one field with the same name (§8.3.3.3). Such a situation does not in itself cause a compile-time error. However, any attempt within the body of the class to refer to any such field by its simple name will result in a compile-time error, because such a reference is ambiguous.

There might be several paths by which the same field declaration might be inherited from an interface. In such a situation, the field is considered to be inherited only once, and it may be referred to by its simple name without ambiguity.

A hidden field can be accessed by using a qualified name (if it is `static`) or by using a field access expression (§15.10) that contains the keyword `super` or a cast to a superclass type. See §15.10.2 for discussion and an example.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.3.1 Field Modifiers

FieldModifiers:

FieldModifier

FieldModifiers FieldModifier

FieldModifier: one of

public protected private

final static transient volatile

The access modifiers `public`, `protected`, and `private` are discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in a field declaration, or if a field declaration has more than one of the access modifiers `public`, `protected`, and `private`. If two or more (distinct) field modifiers appear in a field declaration, it is customary, though not required, that they appear in the order consistent with that shown above in the production for *FieldModifier*.

8.3.1.1 static Fields

If a field is declared `static`, there exists exactly one incarnation of the field, no matter how many instances (possibly zero) of the class may eventually be created. A `static` field, sometimes called a *class variable*, is incarnated when the class is initialized ([§12.4](#)).

A field that is not declared `static` (sometimes called a non-`static` field) is called an *instance variable*. Whenever a new instance of a class is created, a new variable associated with that instance is created for every instance variable declared in that class or any of its superclasses.

The example program:

```
class Point {
    int x, y, useCount;
    Point(int x, int y) { this.x = x; this.y = y; }
    final static Point origin = new Point(0, 0);
}

class Test {
    public static void main(String[] args) {
        Point p = new Point(1,1);
        Point q = new Point(2,2);
        p.x = 3; p.y = 3; p.useCount++; p.origin.useCount++;
        System.out.println("(" + q.x + "," + q.y + ")");
        System.out.println(q.useCount);
        System.out.println(q.origin == Point.origin);
        System.out.println(q.origin.useCount);
    }
}
```



```
}
```

prints:

```
(2,2)
0
true
1
```

showing that changing the fields `x`, `y`, and `useCount` of `p` does not affect the fields of `q`, because these fields are instance variables in distinct objects. In this example, the class variable `origin` of the class `Point` is referenced both using the class name as a qualifier, in `Point.origin`, and using variables of the class type in field access expressions (§15.10), as in `p.origin` and `q.origin`. These two ways of accessing the `origin` class variable access the same object, evidenced by the fact that the value of the reference equality expression (§15.20.3):

```
q.origin==Point.origin
```

is `true`. Further evidence is that the incrementation:

```
p.origin.useCount++;
```

causes the value of `q.origin.useCount` to be 1; this is so because `p.origin` and `q.origin` refer to the same variable.

8.3.1.2 final Fields

A field can be declared `final`, in which case its declarator must include a variable initializer or a compile-time error occurs. Both class and instance variables (`static` and non-`static` fields) may be declared `final`.

Any attempt to assign to a `final` field results in a compile-time error. Therefore, once a `final` field has been initialized, it always contains the same value. If a `final` field holds a reference to an object, then the state of the object may be changed by operations on the object, but the field will always refer to the same object. This applies also to arrays, because arrays are objects; if a `final` field holds a reference to an array, then the components of the array may be changed by operations on the array, but the field will always refer to the same array.

Declaring a field `final` can serve as useful documentation that its value will not change, can help to avoid programming errors, and can make it easier for a compiler to generate efficient code.

In the example:

```
class Point {
    int x, y;
    int useCount;
```



```

    Point(int x, int y) { this.x = x; this.y = y; }
    final static Point origin = new Point(0, 0);
}

```

the class `Point` declares a final class variable `origin`. The `origin` variable holds a reference to an object that is an instance of class `Point` whose coordinates are (0, 0). The value of the variable `Point.origin` can never change, so it always refers to the same `Point` object, the one created by its initializer. However, an operation on this `Point` object might change its state—for example, modifying its `useCount` or even, misleadingly, its `x` or `y` coordinate.

8.3.1.3 transient Fields

Variables may be marked `transient` to indicate that they are not part of the persistent state of an object. If an instance of the class `Point`:

```

class Point {
    int x, y;
    transient float rho, theta;
}

```

were saved to persistent storage by a system service, then only the fields `x` and `y` would be saved. This specification does not yet specify details of such services; we intend to provide them in a future version of this specification.

8.3.1.4 volatile Fields

As described in [§17](#), the Java language allows threads that access shared variables to keep private working copies of the variables; this allows a more efficient implementation of multiple threads. These working copies need be reconciled with the master copies in the shared main memory only at prescribed synchronization points, namely when objects are locked or unlocked. As a rule, to ensure that shared variables are consistently and reliably updated, a thread should ensure that it has exclusive use of such variables by obtaining a lock that, conventionally, enforces mutual exclusion for those shared variables.

Java provides a second mechanism that is more convenient for some purposes: a field may be declared `volatile`, in which case a thread must reconcile its working copy of the field with the master copy every time it accesses the variable. Moreover, operations on the master copies of one or more volatile variables on behalf of a thread are performed by the main memory in exactly the order that the thread requested.

If, in the following example, one thread repeatedly calls the method `one` (but no more than `Integer.MAX_VALUE` [\(§20.7.2\)](#) times in all), and another thread repeatedly calls the method `two`:

```

class Test {
    static int i = 0, j = 0;
}

```



```

static void one() { i++; j++; }

static void two() {
    System.out.println("i=" + i + " j=" + j);
}
}

```

then method `two` could occasionally print a value for `j` that is greater than the value of `i`, because the example includes no synchronization and, under the rules explained in [§17](#), the shared values of `i` and `j` might be updated out of order.

One way to prevent this out-of-order behavior would be to declare methods `one` and `two` to be synchronized ([§8.4.3.5](#)):

```

class Test {

    static int i = 0, j = 0;

    static synchronized void one() { i++; j++; }

    static synchronized void two() {
        System.out.println("i=" + i + " j=" + j);
    }

}

```

This prevents method `one` and method `two` from being executed concurrently, and furthermore guarantees that the shared values of `i` and `j` are both updated before method `one` returns. Therefore method `two` never observes a value for `j` greater than that for `i`; indeed, it always observes the same value for `i` and `j`.

Another approach would be to declare `i` and `j` to be `volatile`:

```

class Test {

    static volatile int i = 0, j = 0;

    static void one() { i++; j++; }

    static void two() {
        System.out.println("i=" + i + " j=" + j);
    }

}

```

This allows method `one` and method `two` to be executed concurrently, but guarantees that accesses to the shared values for `i` and `j` occur exactly as many times, and in exactly the same order, as they

appear to occur during execution of the program text by each thread. Therefore, method `two` never observes a value for `j` greater than that for `i`, because each update to `i` must be reflected in the shared value for `i` before the update to `j` occurs. It is possible, however, that any given invocation of method `two` might observe a value for `j` that is much greater than the value observed for `i`, because method `one` might be executed many times between the moment when method `two` fetches the value of `i` and the moment when method `two` fetches the value of `j`.

See [§17](#) for more discussion and examples.

A compile-time error occurs if a `final` variable is also declared `volatile`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.3.2 Initialization of Fields

If a field declarator contains a *variable initializer*, then it has the semantics of an assignment (§15.25) to the declared variable, and:

- If the declarator is for a class variable (that is, a `static` field), then the variable initializer is evaluated and the assignment performed exactly once, when the class is initialized (§12.4).
- If the declarator is for an instance variable (that is, a field that is not `static`), then the variable initializer is evaluated and the assignment performed each time an instance of the class is created (§12.5).

The example:

```
class Point {
    int x = 1, y = 5;
}

class Test {
    public static void main(String[] args) {
        Point p = new Point();
        System.out.println(p.x + ", " + p.y);
    }
}
```

produces the output:

```
1, 5
```

because the assignments to `x` and `y` occur whenever a new `Point` is created.

Variable initializers are also used in local variable declaration statements (§14.3), where the initializer is evaluated and the assignment performed each time the local variable declaration statement is executed.

It is a compile-time error if the evaluation of a variable initializer for a field of a class (or interface) can complete abruptly with a checked exception (§11.2).

8.3.2.1 Initializers for Class Variables

A compile-time error occurs if an initialization expression for a class variable contains a use by a simple name of that class variable or of another class variable whose declaration occurs to its right (that is, textually later) in the same class. Thus:

```
class Test {
    static float f = j;                                     //
compile-time error: forward reference
    static int j = 1;
    static int k = k+1;                                     //
```



```
compile-time error: forward reference
}
```

causes two compile-time errors, because `j` is referred to in the initialization of `f` before `j` is declared and because the initialization of `k` refers to `k` itself.

If a reference by simple name to any instance variable occurs in an initialization expression for a class variable, then a compile-time error occurs.

If the keyword `this` (§15.7.2) or the keyword `super` (§15.10.2, §15.11) occurs in an initialization expression for a class variable, then a compile-time error occurs.

(One subtlety here is that, at run time, `static` variables that are `final` and that are initialized with compile-time constant values are initialized first. This also applies to such fields in interfaces (§9.3.1). These variables are "constants" that will never be observed to have their default initial values (§4.5.4), even by devious programs. See §12.4.2 and §13.4.8 for more discussion.)

8.3.2.2 Initializers for Instance Variables

A compile-time error occurs if an initialization expression for an instance variable contains a use by a simple name of that instance variable or of another instance variable whose declaration occurs to its right (that is, textually later) in the same class. Thus:

```
class Test {
    float f = j;
    int j = 1;
    int k = k+1;
}
```

causes two compile-time errors, because `j` is referred to in the initialization of `f` before `j` is declared and because the initialization of `k` refers to `k` itself.

Initialization expressions for instance variables may use the simple name of any `static` variable declared in or inherited by the class, even one whose declaration occurs textually later. Thus the example:

```
class Test {
    float f = j;
    static int j = 1;
}
```

compiles without error; it initializes `j` to 1 when class `Test` is initialized, and initializes `f` to the current value of `j` every time an instance of class `Test` is created.

Initialization expressions for instance variables are permitted to refer to the current object `this` (§15.7.2) and to use the keyword `super` (§15.10.2, §15.11).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.3.3 Examples of Field Declarations

The following examples illustrate some (possibly subtle) points about field declarations.

8.3.3.1 Example: Hiding of Class Variables

The example:

```
class Point {
    static int x = 2;
}

class Test extends Point {
    static double x = 4.7;
    public static void main(String[] args) {

        new Test().printX();
    }
    void printX() {
        System.out.println(x + " " + super.x);
    }
}
```

produces the output:

```
4.7 2
```

because the declaration of `x` in class `Test` hides the definition of `x` in class `Point`, so class `Test` does not inherit the field `x` from its superclass `Point`. Within the declaration of class `Test`, the simple name `x` refers to the field declared within class `Test`. Code in class `Test` may refer to the field `x` of class `Point` as `super.x` (or, because `x` is static, as `Point.x`). If the declaration of `Test.x` is deleted:

```
class Point {
    static int x = 2;
}

class Test extends Point {
    public static void main(String[] args) {
        new Test().printX();
    }
    void printX() {
        System.out.println(x + " " + super.x);
    }
}
```


then the field `x` of class `Point` is no longer hidden within class `Test`; instead, the simple name `x` now refers to the field `Point.x`. Code in class `Test` may still refer to that same field as `super.x`. Therefore, the output from this variant program is:

```
2 2
```

8.3.3.2 Example: Hiding of Instance Variables

This example is similar to that in the previous section, but uses instance variables rather than static variables. The code:

```
class Point {
    int x = 2;
}

class Test extends Point {
    double x = 4.7;
    void printBoth() {
        System.out.println(x + " " + super.x);
    }
    public static void main(String[] args) {
        Test sample = new Test();
        sample.printBoth();
        System.out.println(sample.x + " " +

        ((Point) sample).x);
    }
}
```

produces the output:

```
4.7 2
4.7 2
```

because the declaration of `x` in class `Test` hides the definition of `x` in class `Point`, so class `Test` does not inherit the field `x` from its superclass `Point`. It must be noted, however, that while the field `x` of class `Point` is not *inherited* by class `Test`, it is nevertheless *implemented* by instances of class `Test`. In other words, every instance of class `Test` contains two fields, one of type `int` and one of type `float`. Both fields bear the name `x`, but within the declaration of class `Test`, the simple name `x` always refers to the field declared within class `Test`. Code in instance methods of class `Test` may refer to the instance variable `x` of class `Point` as `super.x`.

Code that uses a field access expression to access field `x` will access the field named `x` in the class indicated by the type of reference expression. Thus, the expression `sample.x` accesses a `float`

value, the instance variable declared in class `Test`, because the type of the variable `sample` is `Test`, but the expression `((Point) sample).x` accesses an `int` value, the instance variable declared in class `Point`, because of the cast to type `Point`.

If the declaration of `x` is deleted from class `Test`, as in the program:

```
class Point {
    static int x = 2;
}

class Test extends Point {
    void printBoth() {
        System.out.println(x + " " + super.x);
    }
    public static void main(String[] args) {
        Test sample = new Test();
        sample.printBoth();
        System.out.println(sample.x + " " +

        ((Point) sample).x);
    }
}
```

then the field `x` of class `Point` is no longer hidden within class `Test`. Within instance methods in the declaration of class `Test`, the simple name `x` now refers to the field declared within class `Point`. Code in class `Test` may still refer to that same field as `super.x`. The expression `sample.x` still refers to the field `x` within type `Test`, but that field is now an inherited field, and so refers to the field `x` declared in class `Point`. The output from this variant program is:

```
2 2
2 2
```

8.3.3.3 Example: Multiply Inherited Fields

A class may inherit two or more fields with the same name, either from two interfaces or from its superclass and an interface. A compile-time error occurs on any attempt to refer to any ambiguously inherited field by its simple name. A qualified name or a field access expression that contains the keyword `super` (§15.10.2) may be used to access such fields unambiguously. In the example:

```
interface Frob { float v = 2.0f; }

class SuperTest { int v = 3; }
```



```

class Test extends SuperTest implements Frob {
    public static void main(String[] args) {
        new Test().printV();
    }
    void printV() { System.out.println(v); }
}

```

the class `Test` inherits two fields named `v`, one from its superclass `SuperTest` and one from its superinterface `Frob`. This in itself is permitted, but a compile-time error occurs because of the use of the simple name `v` in method `printV`: it cannot be determined which `v` is intended.

The following variation uses the field access expression `super.v` to refer to the field named `v` declared in class `SuperTest` and uses the qualified name `Frob.v` to refer to the field named `v` declared in interface `Frob`:

```

interface Frob { float v = 2.0f; }

```

```

class SuperTest { int v = 3; }

```

```

class Test extends SuperTest implements Frob {
    public static void main(String[] args) {
        new Test().printV();
    }
    void printV() {
        System.out.println((super.v + Frob.v)/2);
    }
}

```

It compiles and prints:

2.5

Even if two distinct inherited fields have the same type, the same value, and are both `final`, any reference to either field by simple name is considered ambiguous and results in a compile-time error. In the example:

```

interface Color { int RED=0, GREEN=1, BLUE=2; }

```

```

interface TrafficLight { int RED=0, YELLOW=1, GREEN=2; }

```



```

class Test implements Color, TrafficLight {
    public static void main(String[] args) {
        System.out.println(GREEN);
        // compile-time error
        System.out.println(RED);
        // compile-time error
    }
}

```

it is not astonishing that the reference to `GREEN` should be considered ambiguous, because class `Test` inherits two different declarations for `GREEN` with different values. The point of this example is that the reference to `RED` is also considered ambiguous, because two distinct declarations are inherited. The fact that the two fields named `RED` happen to have the same type and the same unchanging value does not affect this judgment.

8.3.3.4 Example: Re-inheritance of Fields

If the same field declaration is inherited from an interface by multiple paths, the field is considered to be inherited only once. It may be referred to by its simple name without ambiguity. For example, in the code:

```

public interface Colorable {
    int RED = 0xff0000, GREEN = 0x00ff00, BLUE = 0x0000ff;
}

public interface Paintable extends Colorable {
    int MATTE = 0, GLOSSY = 1;
}

class Point { int x, y; }

class ColoredPoint extends Point implements Colorable {
    . . .
}

class PaintedPoint extends ColoredPoint implements Paintable
{
    . . .      RED      . . .
}

```

the fields `RED`, `GREEN`, and `BLUE` are inherited by the class `PaintedPoint` both through its direct superclass `ColoredPoint` and through its direct superinterface `Paintable`. The simple names `RED`, `GREEN`, and `BLUE` may nevertheless be used without ambiguity within the class `PaintedPoint` to refer to the fields declared in interface `Colorable`.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4 Method Declarations

The diversity of physical arguments and opinions embraces all sorts of methods.

--Michael de Montaigne (1533-1592), *Of Experience*

A *method* declares executable code that can be invoked, passing a fixed number of values as arguments.

MethodDeclaration:

MethodHeader MethodBody

MethodHeader:

MethodModifiersopt ResultType MethodDeclarator Throwsopt

ResultType:

Type

void

MethodDeclarator:

Identifier (FormalParameterListopt)

The *MethodModifiers* are described in [§8.4.3](#), the *Throws* clause in [§8.4.4](#), and the *MethodBody* in [§8.4.5](#). A method declaration either specifies the type of value that the method returns or uses the keyword `void` to indicate that the method does not return a value.

The *Identifier* in a *MethodDeclarator* may be used in a name to refer to the method. A class can declare a method with the same name as the class or a field of the class.

For compatibility with older versions of Java, a declaration form for a method that returns an array is allowed to place (some or all of) the empty bracket pairs that form the declaration of the array type after the parameter list. This is supported by the obsolescent production:

MethodDeclarator:

MethodDeclarator []

but should not be used in new Java code.

It is a compile-time error for the body of a class to have as members two methods with the same signature ([§8.4.2](#)) (name, number of parameters, and types of any parameters). Methods and fields may have the same name, since they are used in different contexts and are disambiguated by the different lookup procedures ([§6.5](#)).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.1 Formal Parameters

The *formal parameters* of a method, if any, are specified by a list of comma-separated parameter specifiers. Each parameter specifier consists of a type and an identifier (optionally followed by brackets) that specifies the name of the parameter:

FormalParameterList:

FormalParameter

FormalParameterList , *FormalParameter*

FormalParameter:

Type *VariableDeclaratorId*

The following is repeated from §8.3 to make the presentation here clearer:

VariableDeclaratorId:

Identifier

VariableDeclaratorId []

If a method has no parameters, only an empty pair of parentheses appears in the method's declaration.

If two formal parameters are declared to have the same name (that is, their declarations mention the same *Identifier*), then a compile-time error occurs.

When the method is invoked (§15.11), the values of the actual argument expressions initialize newly created parameter variables, each of the declared *Type*, before execution of the body of the method. The *Identifier* that appears in the *DeclaratorId* may be used as a simple name in the body of the method to refer to the formal parameter.

The scope of formal parameter names is the entire body of the method. These parameter names may not be redeclared as local variables or exception parameters within the method; that is, hiding the name of a parameter is not permitted.

Formal parameters are referred to only using simple names, never by using qualified names (§6.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.2 Method Signature

The *signature* of a method consists of the name of the method and the number and types of formal parameters to the method. A class may not declare two methods with the same signature, or a compile-time error occurs. The example:

```
class Point implements Move {  
    int x, y;  
    abstract void move(int dx, int dy);  
    void move(int dx, int dy) { x += dx; y += dy; }  
}
```

causes a compile-time error because it declares two `move` methods with the same signature. This is an error even though one of the declarations is `abstract`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.3 Method Modifiers

MethodModifiers:

MethodModifier

MethodModifiers MethodModifier

MethodModifier: one of

public protected private

abstract static final synchronized native

The access modifiers `public`, `protected`, and `private` are discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in a method declaration, or if a method declaration has more than one of the access modifiers `public`, `protected`, and `private`. A compile-time error occurs if a method declaration that contains the keyword `abstract` also contains any one of the keywords `private`, `static`, `final`, `native`, or `synchronized`.

If two or more method modifiers appear in a method declaration, it is customary, though not required, that they appear in the order consistent with that shown above in the production for *MethodModifier*.

8.4.3.1 abstract Methods

An `abstract` method declaration introduces the method as a member, providing its signature (name and number and type of parameters), return type, and `throws` clause (if any), but does not provide an implementation. The declaration of an `abstract` method *m* must appear within an `abstract` class (call it *A*); otherwise a compile-time error results. Every subclass of *A* that is not `abstract` must provide an implementation for *m*, or a compile-time error occurs. More precisely, for every subclass *C* of the `abstract` class *A*, if *C* is not `abstract`, then there must be some class *B* such that all of the following are true:

- *B* is a superclass of *C* or is *C* itself.
- *B* is a subclass of *A*.
- *B* provides a declaration of the method *m* that is not `abstract`, and this declaration is inherited by *C*, thereby providing an implementation of method *m* that is visible to *C*.

If there is no such class *B*, then a compile-time error occurs.

It is a compile-time error for a `private` method to be declared `abstract`. It would be impossible for a subclass to implement a `private abstract` method, because `private` methods are not visible to subclasses; therefore such a method could never be used.

It is a compile-time error for a `static` method to be declared `abstract`.

It is a compile-time error for a `final` method to be declared `abstract`.

An `abstract` class can override an `abstract` method by providing another `abstract` method declaration. This can provide a place to put a documentation comment ([§18](#)), or to declare that the set of checked exceptions ([§11.2](#)) that can be thrown by that method, when it is implemented by its

subclasses, is to be more limited. For example, consider this code:

```
class BufferEmpty extends Exception {
    BufferEmpty() { super(); }
    BufferEmpty(String s) { super(s); }
}

class BufferError extends Exception {
    BufferError() { super(); }
    BufferError(String s) { super(s); }
}

public interface Buffer {
    char get() throws BufferEmpty, BufferError;
}

public abstract class InfiniteBuffer implements Buffer {
    abstract char get() throws BufferError;
}
```

The overriding declaration of method `get` in class `InfiniteBuffer` states that method `get` in any subclass of `InfiniteBuffer` never throws a `BufferEmpty` exception, putatively because it generates the data in the buffer, and thus can never run out of data.

An instance method that is not abstract can be overridden by an abstract method. For example, we can declare an abstract class `Point` that requires its subclasses to implement `toString` if they are to be complete, instantiable classes:

```
abstract class Point {
    int x, y;
    public abstract String toString();
}
```

This abstract declaration of `toString` overrides the non-abstract `toString` method of class `Object` ([§20.1.2](#)). (Class `Object` is the implicit direct superclass of class `Point`.) Adding the code:

```
class ColoredPoint extends Point {
    int color;
    public String toString() {
        return super.toString() + ": color " + color; // error
    }
}
```

results in a compile-time error because the invocation `super.toString()` refers to method

`toString` in class `Point`, which is `abstract` and therefore cannot be invoked. Method `toString` of class `Object` can be made available to class `ColoredPoint` only if class `Point` explicitly makes it available through some other method, as in:

```
abstract class Point {
    int x, y;
    public abstract String toString();
    protected String objString() { return super.toString(); }
}

class ColoredPoint extends Point {
    int color;
    public String toString() {
        return objString() + ": color " + color;
    }
}
```

// correct

8.4.3.2 static Methods

A method that is declared `static` is called a *class method*. A class method is always invoked without reference to a particular object. An attempt to reference the current object using the keyword `this` or the keyword `super` in the body of a class method results in a compile time error. It is a compile-time error for a `static` method to be declared `abstract`.

A method that is not declared `static` is called an *instance method*, and sometimes called a non-`static` method). An instance method is always invoked with respect to an object, which becomes the current object to which the keywords `this` and `super` refer during execution of the method body.

8.4.3.3 final Methods

A method can be declared `final` to prevent subclasses from overriding or hiding it. It is a compile-time error to attempt to override or hide a `final` method.

A `private` method and all methods declared in a `final` class ([§8.1.2.2](#)) are implicitly `final`, because it is impossible to override them. It is permitted but not required for the declarations of such methods to redundantly include the `final` keyword.

It is a compile-time error for a `final` method to be declared `abstract`.

At run-time, a machine-code generator or optimizer can easily and safely "inline" the body of a `final` method, replacing an invocation of the method with the code in its body, as in the example:

```
final class Point {
    int x, y;
    void move(int dx, int dy) { x += dx; y += dy; }
}
```



```

class Test {
    public static void main(String[] args) {
        Point[] p = new Point[100];
        for (int i = 0; i < p.length; i++) {
            p[i] = new Point();
            p[i].move(i, p.length-1-i);
        }
    }
}

```

Here, inlining the method `move` of class `Point` in method `main` would transform the `for` loop to the form:

```

        for (int i = 0; i < p.length; i++) {
            p[i] = new Point();
            Point pi = p[i];
            pi.x += i;
            pi.y += p.length-1-i;
        }

```

The loop might then be subject to further optimizations.

Such inlining cannot be done at compile time unless it can be guaranteed that `Test` and `Point` will always be recompiled together, so that whenever `Point`-and specifically its `move` method-changes, the code for `Test.main` will also be updated.

8.4.3.4 native Methods

A method that is `native` is implemented in platform-dependent code, typically written in another programming language such as C, C++, FORTRAN, or assembly language. The body of a `native` method is given as a semicolon only, indicating that the implementation is omitted, instead of a block.

A compile-time error occurs if a `native` method is declared `abstract`.

For example, the class `RandomAccessFile` of the standard package `java.io` might declare the following `native` methods:

```

package java.io;

```

```

public class RandomAccessFile
    implements DataOutput, DataInput
{
    . . .
    public native void open(String name, boolean writeable)
        throws IOException;
}

```



```

    public native int readBytes(byte[] b, int off, int len)
        throws IOException;
    public native void writeBytes(byte[] b, int off, int len)
        throws IOException;
    public native long getFilePointer() throws IOException;
    public native void seek(long pos) throws IOException;
    public native long length() throws IOException;
    public native void close() throws IOException;
}

```

8.4.3.5 synchronized Methods

A synchronized method acquires a lock ([§17.1](#)) before it executes. For a class (static) method, the lock associated with the `Class` object ([§20.3](#)) for the method's class is used. For an instance method, the lock associated with `this` (the object for which the method was invoked) is used. These are the same locks that can be used by the `synchronized` statement ([§14.17](#)); thus, the code:

```

class Test {
    int count;
    synchronized void bump() { count++; }
    static int classCount;
    static synchronized void classBump() {
        classCount++;
    }
}

```

has exactly the same effect as:

```

class BumpTest {
    int count;
    void bump() {
        synchronized (this) {
            count++;
        }
    }
    static int classCount;
    static void classBump() {
        try {
            synchronized (Class.forName("BumpTest")) {
                classCount++;
            }
        } catch (ClassNotFoundException e) {
            ...
        }
    }
}

```


The more elaborate example:

```
public class Box {

    public synchronized Object get() {
        return contents;
    }

    public synchronized boolean put(Object contents) {
        return false;
        return true;
    }

}
```

defines a class which is designed for concurrent use. Each instance of the class `Box` has an instance variable `contents` that can hold a reference to any object. You can put an object in a `Box` by invoking `put`, which returns `false` if the box is already full. You can get something out of a `Box` by invoking `get`, which returns a null reference if the `box` is empty.

If `put` and `get` were not `synchronized`, and two threads were executing methods for the same instance of `Box` at the same time, then the code could misbehave. It might, for example, lose track of an object because two invocations to `put` occurred at the same time.

See [§17](#) for more discussion of threads and locks.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.4 Throws

A *throws clause* is used to declare any checked exceptions (§11.2) that can result from the execution of a method or constructor:

Throws:

throws ClassTypeList

ClassTypeList:

ClassType

ClassTypeList , *ClassType*

A compile-time error occurs if any *ClassType* mentioned in a *throws* clause is not the class `Throwable` (§20.22) or a subclass of `Throwable`. It is permitted but not required to mention other (unchecked) exceptions in a *throws* clause.

For each checked exception that can result from execution of the body of a method or constructor, a compile-time error occurs unless that exception type or a superclass of that exception type is mentioned in a *throws* clause in the declaration of the method or constructor.

The requirement to declare checked exceptions allows the compiler to ensure that code for handling such error conditions has been included. Methods or constructors that fail to handle exceptional conditions thrown as checked exceptions will normally result in a compile-time error because of the lack of a proper exception type in a *throws* clause. Java thus encourages a programming style where rare and otherwise truly exceptional conditions are documented in this way.

The predefined exceptions that are not checked in this way are those for which declaring every possible occurrence would be unimaginably inconvenient:

- Exceptions that are represented by the subclasses of class `Error`, for example `OutOfMemoryError`, are thrown due to a failure in or of the virtual machine. Many of these are the result of linkage failures and can occur at unpredictable points in the execution of a Java program. Sophisticated programs may yet wish to catch and attempt to recover from some of these conditions.
- The exceptions that are represented by the subclasses of the class `RuntimeException`, for example `NullPointerException`, result from runtime integrity checks and are thrown either directly from the Java program or in library routines. It is beyond the scope of the Java language, and perhaps beyond the state of the art, to include sufficient information in the program to reduce to a manageable number the places where these can be proven not to occur.

A method that overrides or hides another method (§8.4.6), including methods that implement abstract methods defined in interfaces, may not be declared to throw more checked exceptions than the overridden or hidden method.

More precisely, suppose that *B* is a class or interface, and *A* is a superclass or superinterface of *B*, and a method declaration *n* in *B* overrides or hides a method declaration *m* in *A*. If *n* has a *throws* clause that mentions any checked exception types, then *m* must have a *throws* clause, and for every checked exception type listed in the *throws* clause of *n*, that same exception class or one of its superclasses must occur in the *throws* clause of *m*; otherwise, a compile-time error occurs.

See §11 for more information about exceptions and a large example.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.5 Method Body

A *method body* is either a block of code that implements the method or simply a semicolon, indicating the lack of an implementation. The body of a method must be a semicolon if and only if the method is either `abstract` (§8.4.3.1) or `native` (§8.4.3.4).

MethodBody:

Block

;

A compile-time error occurs if a method declaration is either `abstract` or `native` and has a block for its body. A compile-time error occurs if a method declaration is neither `abstract` nor `native` and has a semicolon for its body.

If an implementation is to be provided for a method but the implementation requires no executable code, the method body should be written as a block that contains no statements: "{ }".

If a method is declared `void`, then its body must not contain any `return` statement (§14.15) that has an *Expression*.

If a method is declared to have a return type, then every `return` statement (§14.15) in its body must have an *Expression*. A compile-time error occurs if the body of the method can complete normally (§14.1). In other words, a method with a return type must return only by using a `return` statement that provides a value `return`; it is not allowed to "drop off the end of its body."

Note that it is possible for a method to have a declared return type and yet contain no `return` statements. Here is one example:

```
class DizzyDean {  
    int pitch() { throw new RuntimeException("90 mph?!"); }  
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

8.4.6 Inheritance, Overriding, and Hiding

A class *inherits* from its direct superclass and direct superinterfaces all the methods (whether `abstract` or not) of the superclass and superinterfaces that are accessible to code in the class and are neither overridden ([§8.4.6.1](#)) nor hidden ([§8.4.6.2](#)) by a declaration in the class.

8.4.6.1 Overriding (By Instance Methods)

If a class declares an instance method, then the declaration of that method is said to *override* any and all methods with the same signature in the superclasses and superinterfaces of the class that would otherwise be accessible to code in the class. Moreover, if the method declared in the class is not `abstract`, then the declaration of that method is said to *implement* any and all declarations of `abstract` methods with the same signature in the superclasses and superinterfaces of the class that would otherwise be accessible to code in the class.

A compile-time error occurs if an instance method overrides a `static` method. In this respect, overriding of methods differs from hiding of fields ([§8.3](#)), for it is permissible for an instance variable to hide a `static` variable.

An overridden method can be accessed by using a method invocation expression ([§15.11](#)) that contains the keyword `super`. Note that a qualified name or a cast to a superclass type is not effective in attempting to access an overridden method; in this respect, overriding of methods differs from hiding of fields. See [§15.11.4.10](#) for discussion and examples of this point.

8.4.6.2 Hiding (By Class Methods)

If a class declares a `static` method, then the declaration of that method is said to *hide* any and all methods with the same signature in the superclasses and superinterfaces of the class that would otherwise be accessible to code in the class. A compile-time error occurs if a `static` method hides an instance method. In this respect, hiding of methods differs from hiding of fields ([§8.3](#)), for it is permissible for a `static` variable to hide an instance variable.

A hidden method can be accessed by using a qualified name or by using a method invocation expression ([§15.11](#)) that contains the keyword `super` or a cast to a superclass type. In this respect, hiding of methods is similar to hiding of fields.

8.4.6.3 Requirements in Overriding and Hiding

If a method declaration overrides or hides the declaration of another method, then a compile-time error occurs if they have different return types or if one has a return type and the other is `void`. Moreover, a method declaration must not have a `throws` clause that conflicts ([§8.4.4](#)) with that of any method that it overrides or hides; otherwise, a compile-time error occurs. In these respects, overriding of methods differs from hiding of fields ([§8.3](#)), for it is permissible for a field to hide a field of another type.

The access modifier ([§6.6](#)) of an overriding or hiding method must provide at least as much access as the overridden or hidden method, or a compile-time error occurs. In more detail:

- If the overridden or hidden method is `public`, then the overriding or hiding method must be `public`; otherwise, a compile-time error occurs.

- If the overridden or hidden method is `protected`, then the overriding or hiding method must be `protected` or `public`; otherwise, a compile-time error occurs.
- If the overridden or hidden method has default (package) access, then the overriding or hiding method must not be `private`; otherwise, a compile-time error occurs.

Note that a `private` method is never accessible to subclasses and so cannot be hidden or overridden in the technical sense of those terms. This means that a subclass can declare a method with the same signature as a `private` method in one of its superclasses, and there is no requirement that the return type or `throws` clause of such a method bear any relationship to those of the `private` method in the superclass.

8.4.6.4 Inheriting Methods with the Same Signature

It is possible for a class to inherit more than one method with the same signature (§8.4.6.4). Such a situation does not in itself cause a compile-time error. There are then two possible cases:

- If one of the inherited methods is not `abstract`, then there are two subcases:
 - If the method that is not `abstract` is `static`, a compile-time error occurs.
 - Otherwise, the method that is not `abstract` is considered to override, and therefore to implement, all the other methods on behalf of the class that inherits it. A compile-time error occurs if, comparing the method that is not `abstract` with each of the other of the inherited methods, for any such pair, either they have different return types or one has a return type and the other is `void`. Moreover, a compile-time error occurs if the inherited method that is not `abstract` has a `throws` clause that conflicts (§8.4.4) with that of any other of the inherited methods.
- If none of the inherited methods is not `abstract`, then the class is necessarily an `abstract` class and is considered to inherit all the `abstract` methods. A compile-time error occurs if, for any two such inherited methods, either they have different return types or one has a return type and the other is `void`. (The `throws` clauses do not cause errors in this case.)

It is not possible for two or more inherited methods with the same signature not to be `abstract`, because methods that are not `abstract` are inherited only from the direct superclass, not from superinterfaces.

There might be several paths by which the same method declaration might be inherited from an interface. This fact causes no difficulty and never, of itself, results in a compile-time error.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.7 Overloading

If two methods of a class (whether both declared in the same class, or both inherited by a class, or one declared and one inherited) have the same name but different signatures, then the method name is said to be *overloaded*. This fact causes no difficulty and never of itself results in a compile-time error. There is no required relationship between the return types or between the `throws` clauses of two methods with the same name but different signatures.

Methods are overridden on a signature-by-signature basis. If, for example, a class declares two `public` methods with the same name, and a subclass overrides one of them, the subclass still inherits the other method. In this respect, Java differs from C++.

When a method is invoked ([§15.11](#)), the number of actual arguments and the compile-time types of the arguments are used, at compile time, to determine the signature of the method that will be invoked ([§15.11.2](#)). If the method that is to be invoked is an instance method, the actual method to be invoked will be determined at run time, using dynamic method lookup ([§15.11.4](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.4.8 Examples of Method Declarations

The following examples illustrate some (possibly subtle) points about method declarations.

8.4.8.1 Example: Overriding

In the example:

```
class Point {
    int x = 0, y = 0;

    void move(int dx, int dy) { x += dx; y += dy; }
}

class SlowPoint extends Point {
    int xLimit, yLimit;

    void move(int dx, int dy) {
        super.move(limit(dx, xLimit), limit(dy, yLimit));
    }

    static int limit(int d, int limit) {
        return d > limit ? limit : d < -limit ? -limit : d;
    }
}
```

the class `SlowPoint` overrides the declarations of method `move` of class `Point` with its own `move` method, which limits the distance that the point can move on each invocation of the method. When the `move` method is invoked for an instance of class `SlowPoint`, the overriding definition in class `SlowPoint` will always be called, even if the reference to the `SlowPoint` object is taken from a variable whose type is `Point`.

8.4.8.2 Example: Overloading, Overriding, and Hiding

In the example:

```
class Point {
    int x = 0, y = 0;
```



```

        void move(int dx, int dy) { x += dx; y += dy; }

        int color;
    }

    class RealPoint extends Point {

        float x = 0.0f, y = 0.0f;

        void move(int dx, int dy) { move((float)dx, (float)dy); }

        void move(float dx, float dy) { x += dx; y += dy; }
    }

```

the class `RealPoint` hides the declarations of the `int` instance variables `x` and `y` of class `Point` with its own `float` instance variables `x` and `y`, and overrides the method `move` of class `Point` with its own `move` method. It also overloads the name `move` with another method with a different signature (§8.4.2).

In this example, the members of the class `RealPoint` include the instance variable `color` inherited from the class `Point`, the `float` instance variables `x` and `y` declared in `RealPoint`, and the two `move` methods declared in `RealPoint`.

Which of these overloaded `move` methods of class `RealPoint` will be chosen for any particular method invocation will be determined at compile time by the overloading resolution procedure described in §15.11.

8.4.8.3 Example: Incorrect Overriding

This example is an extended variation of that in the preceding section:

```

class Point {

    int x = 0, y = 0, color;

    void move(int dx, int dy) { x += dx; y += dy; }

    int getX() { return x; }

    int getY() { return y; }
}

```



```

class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;

    void move(int dx, int dy) { move((float)dx, (float)dy); }

    void move(float dx, float dy) { x += dx; y += dy; }

    float getX() { return x; }

    float getY() { return y; }
}

```

Here the class `Point` provides methods `getX` and `getY` that return the values of its fields `x` and `y`; the class `RealPoint` then overrides these methods by declaring methods with the same signature. The result is two errors at compile time, one for each method, because the return types do not match; the methods in class `Point` return values of type `int`, but the wanna-be overriding methods in class `RealPoint` return values of type `float`.

8.4.8.4 Example: Overriding versus Hiding

This example corrects the errors of the example in the preceding section:

```

class Point {
    int x = 0, y = 0;

    void move(int dx, int dy) { x += dx; y += dy; }

    int getX() { return x; }

    int getY() { return y; }

    int color;
}

class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;
}

```



```

    void move(int dx, int dy) { move((float)dx, (float)dy); }

    void move(float dx, float dy) { x += dx; y += dy; }

    int getX() { return (int)Math.floor(x); }

    int getY() { return (int)Math.floor(y); }
}

```

Here the overriding methods `getX` and `getY` in class `RealPoint` have the same return types as the methods of class `Point` that they override, so this code can be successfully compiled.

Consider, then, this test program:

```

class Test {

    public static void main(String[] args) {
        RealPoint rp = new RealPoint();
        Point p = rp;
        rp.move(1.71828f, 4.14159f);
        p.move(1, -1);
        show(p.x, p.y);
        show(rp.x, rp.y);
        show(p.getX(), p.getY());
        show(rp.getX(), rp.getY());
    }

    static void show(int x, int y) {
        System.out.println("(" + x + ", " + y + ")");
    }

    static void show(float x, float y) {
        System.out.println("(" + x + ", " + y + ")");
    }
}

```

The output from this program is:

```

(0, 0)
(2.7182798, 3.14159)
(2, 3)
(2, 3)

```


The first line of output illustrates the fact that an instance of `RealPoint` actually contains the two integer fields declared in class `Point`; it is just that their names are hidden from code that occurs within the declaration of class `RealPoint` (and those of any subclasses it might have). When a reference to an instance of class `RealPoint` in a variable of type `Point` is used to access the field `x`, the integer field `x` declared in class `Point` is accessed. The fact that its value is zero indicates that the method invocation `p.move(1, -1)` did not invoke the method `move` of class `Point`; instead, it invoked the overriding method `move` of class `RealPoint`.

The second line of output shows that the field access `rp.x` refers to the field `x` declared in class `RealPoint`. This field is of type `float`, and this second line of output accordingly displays floating-point values. Incidentally, this also illustrates the fact that the method name `show` is overloaded; the types of the arguments in the method invocation dictate which of the two definitions will be invoked.

The last two lines of output show that the method invocations `p.getX()` and `rp.getX()` each invoke the `getX` method declared in class `RealPoint`. Indeed, there is no way to invoke the `getX` method of class `Point` for an instance of class `RealPoint` from outside the body of `RealPoint`, no matter what the type of the variable we may use to hold the reference to the object. Thus, we see that fields and methods behave differently: hiding is different from overriding.

8.4.8.5 Example: Invocation of Hidden Class Methods

A hidden class (`static`) method can be invoked by using a reference whose type is the class that actually contains the declaration of the method. In this respect, hiding of static methods is different from overriding of instance methods. The example:

```
class Super {
    static String greeting() { return "Goodnight"; }
    String name() { return "Richard"; }
}

class Sub extends Super {
    static String greeting() { return "Hello"; }
    String name() { return "Dick"; }
}

class Test {
    public static void main(String[] args) {
        Super s = new Sub();
        System.out.println(s.greeting() + ", " + s.name());
    }
}
```

produces the output:

Goodnight, Dick

because the invocation of `greeting` uses the type of `s`, namely `Super`, to figure out, at compile time, which class method to invoke, whereas the invocation of `name` uses the class of `s`, namely `Sub`, to figure out, at run time, which instance method to invoke.

8.4.8.6 Large Example of Overriding

Overriding makes it easy for subclasses to extend the behavior of an existing class, as shown in this example:

```
import java.io.OutputStream;

import java.io.IOException;

class BufferOutput {

    private OutputStream o;

    BufferOutput(OutputStream o) { this.o = o; }

    protected byte[] buf = new byte[512];

    protected int pos = 0;

    public void putchar(char c) throws IOException {
        if (pos == buf.length)
            flush();
        buf[pos++] = (byte)c;
    }

    public void putstr(String s) throws IOException {
        for (int i = 0; i < s.length(); i++)
            putchar(s.charAt(i));
    }

    public void flush() throws IOException {
        o.write(buf, 0, pos);
        pos = 0;
    }
}
```



```

    }
}

class LineBufferOutput extends BufferOutput {

    LineBufferOutput(OutputStream o) { super(o); }

    public void putchar(char c) throws IOException {
        super.putchar(c);
        if (c == '\n')
            flush();
    }
}

class Test {
    public static void main(String[] args)

        throws IOException

    {
        LineBufferOutput lbo =

            new LineBufferOutput(System.out);
        lbo.putstr("lbo\nlbo");
        System.out.print("print\n");
        lbo.putstr("\n");
    }
}

```

This example produces the output:

```

lbo
print
lbo

```

The class `BufferOutput` implements a very simple buffered version of an `OutputStream`, flushing the output when the buffer is full or `flush` is invoked. The subclass `LineBufferOutput` declares only a constructor and a single method `putchar`, which overrides the method `putchar` of `BufferOutput`. It inherits the methods `putstr` and `flush` from class `Buffer`.

In the `putchar` method of a `LineBufferOutput` object, if the character argument is a newline, then it invokes the `flush` method. The critical point about overriding in this example is that the method `putstr`, which is declared in class `BufferOutput`, invokes the `putchar` method defined by the current object `this`, which is not necessarily the `putchar` method declared in class `BufferOutput`.

Thus, when `putstr` is invoked in `main` using the `LineBufferOutput` object `lbo`, the invocation of `putchar` in the body of the `putstr` method is an invocation of the `putchar` of the object `lbo`, the

overriding declaration of `putchar` that checks for a newline. This allows a subclass of `BufferOutput` to change the behavior of the `putstr` method without redefining it.

Documentation for a class such as `BufferOutput`, which is designed to be extended, should clearly indicate what is the contract between the class and its subclasses, and should clearly indicate that subclasses may override the `putchar` method in this way. The implementor of the `BufferOutput` class would not, therefore, want to change the implementation of `putstr` in a future implementation of `BufferOutput` not to use the method `putchar`, because this would break the preexisting contract with subclasses. See the further discussion of binary compatibility in [§13](#), especially [§13.2](#).

8.4.8.7 Example: Incorrect Overriding because of Throws

This example uses the usual and conventional form for declaring a new exception type, in its declaration of the class `BadPointException`:

```
class BadPointException extends Exception {
    BadPointException() { super(); }
    BadPointException(String s) { super(s); }
}

class Point {
    int x, y;
    void move(int dx, int dy) { x += dx; y += dy; }
}

class CheckedPoint extends Point {
    void move(int dx, int dy) throws BadPointException {
        if ((x + dx) < 0 || (y + dy) < 0)
            throw new BadPointException();
        x += dx; y += dy;
    }
}
```

This example results in a compile-time error, because the override of method `move` in class `CheckedPoint` declares that it will throw a checked exception that the `move` in class `Point` has not declared. If this were not considered an error, an invoker of the method `move` on a reference of type `Point` could find the contract between it and `Point` broken if this exception were thrown.

Removing the `throws` clause does not help:

```
class CheckedPoint extends Point {
    void move(int dx, int dy) {
        if ((x + dx) < 0 || (y + dy) < 0)
            throw new BadPointException();
        x += dx; y += dy;
    }
}
```


A different compile-time error now occurs, because the body of the method `move` cannot throw a checked exception, namely `BadPointException`, that does not appear in the `throws` clause for `move`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.5 Static Initializers

Any *static initializers* declared in a class are executed when the class is initialized and, together with any field initializers (§8.3.2) for class variables, may be used to initialize the class variables of the class (§12.4).

StaticInitializer:

static Block

It is a compile-time error for a static initializer to be able to complete abruptly (§14.1, §15.5) with a checked exception (§11.2).

The static initializers and class variable initializers are executed in textual order and may not refer to class variables declared in the class whose declarations appear textually after the use, even though these class variables are in scope. This restriction is designed to catch, at compile time, circular or otherwise malformed initializations. Thus, both:

```
class Z {
    static int i = j + 2;
    static int j = 4;
}
```

and:

```
class Z {
    static { i = j + 2; }
    static int i, j;
    static { j = 4; }
}
```

result in compile-time errors.

Accesses to class variables by methods are not checked in this way, so:

```
class Z {
    static int peek() { return j; }

    static int i = peek();
    static int j = 1;
}
```

```
class Test {
    public static void main(String[] args) {
```



```
        System.out.println(Z.i);  
    }  
}
```

produces the output:

0

because the variable initializer for `i` uses the class method `peek` to access the value of the variable `j` before `j` has been initialized by its variable initializer, at which point it still has its default value ([§4.5.4](#)).

If a `return` statement ([§14.15](#)) appears anywhere within a static initializer, then a compile-time error occurs.

If the keyword `this` ([§15.7.2](#)) or the keyword `super` ([§15.10](#), [§15.11](#)) appears anywhere within a static initializer, then a compile-time error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6 Constructor Declarations

*The constructor of wharves, bridges, piers, bulk-heads,
floats, stays against the sea . . .*

--Walt Whitman, *Song of the Broad-Axe* (1856)

A *constructor* is used in the creation of an object that is an instance of a class:

ConstructorDeclaration:

ConstructorModifiersopt ConstructorDeclarator

Throwsopt ConstructorBody

ConstructorDeclarator:

SimpleTypeName (FormalParameterListopt)

The *SimpleTypeName* in the *ConstructorDeclarator* must be the simple name of the class that contains the constructor declaration; otherwise a compile-time error occurs. In all other respects, the constructor declaration looks just like a method declaration that has no result type.

Here is a simple example:

```
class Point {  
    int x, y;  
    Point(int x, int y) { this.x = x; this.y = y; }  
}
```

Constructors are invoked by class instance creation expressions (§15.8), by the `newInstance` method of class `Class` (§20.3), by the conversions and concatenations caused by the string concatenation operator `+` (§15.17.1), and by explicit constructor invocations from other constructors (§8.6.5). Constructors are never invoked by method invocation expressions (§15.11).

Access to constructors is governed by access modifiers (§6.6). This is useful, for example, in preventing instantiation by declaring an inaccessible constructor (§8.6.8).

Constructor declarations are not members. They are never inherited and therefore are not subject to hiding or overriding.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.1 Formal Parameters

The formal parameters of a constructor are identical in structure and behavior to the formal parameters of a method ([§8.4.1](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.2 Constructor Signature

The signature of a constructor is identical in structure and behavior to the signature of a method ([§8.4.2](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.3 Constructor Modifiers

ConstructorModifiers:

ConstructorModifier

ConstructorModifiers ConstructorModifier

ConstructorModifier: one of

public protected private

The access modifiers `public`, `protected`, and `private` are discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in a constructor declaration, or if a constructor declaration has more than one of the access modifiers `public`, `protected`, and `private`.

Unlike methods, a constructor cannot be `abstract`, `static`, `final`, `native`, or `synchronized`. A constructor is not inherited, so there is no need to declare it `final` and an `abstract` constructor could never be implemented. A constructor is always invoked with respect to an object, so it makes no sense for a constructor to be `static`. There is no practical need for a constructor to be `synchronized`, because it would lock the object under construction, which is normally not made available to other threads until all constructors for the object have completed their work. The lack of `native` constructors is an arbitrary language design choice that makes it easy for an implementation of the Java Virtual Machine to verify that superclass constructors are always properly invoked during object creation.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

8.6.4 Throws

The `throws` clause for a constructor is identical in structure and behavior to the `throws` clause for a method ([§8.4.4](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.5 Constructor Body

The first statement of a constructor body may be an explicit invocation of another constructor of the same class, written as `this` followed by a parenthesized argument list, or an explicit invocation of a constructor of the direct superclass, written as `super` followed by a parenthesized argument list.

ConstructorBody:

```
{ ExplicitConstructorInvocationopt BlockStatementsopt }
```

ExplicitConstructorInvocation:

```
this ( ArgumentListopt ) ;
```

```
super ( ArgumentListopt ) ;
```

It is a compile-time error for a constructor to directly or indirectly invoke itself through a series of one or more explicit constructor invocations involving `this`.

If a constructor body does not begin with an explicit constructor invocation and the constructor being declared is not part of the primordial class `Object`, then the constructor body is implicitly assumed by the compiler to begin with a superclass constructor invocation "`super()`";, an invocation of the constructor of its direct superclass that takes no arguments.

Except for the possibility of explicit constructor invocations, the body of a constructor is like the body of a method ([§8.4.5](#)). A `return` statement ([§14.15](#)) may be used in the body of a constructor if it does not include an expression.

In the example:

```
class Point {  
    int x, y;  
  
    Point(int x, int y) { this.x = x; this.y = y; }  
}
```

```
class ColoredPoint extends Point {  
    static final int WHITE = 0, BLACK = 1;  
  
    int color;  
  
    ColoredPoint(int x, int y) {  
        this(x, y, WHITE);  
    }  
}
```



```

        ColoredPoint(int x, int y, int color) {
            super(x, y);

            this.color = color;
        }
    }
}

```

the first constructor of `ColoredPoint` invokes the second, providing an additional argument; the second constructor of `ColoredPoint` invokes the constructor of its superclass `Point`, passing along the coordinates.

An explicit constructor invocation statement may not refer to any instance variables or instance methods declared in this class or any superclass, or use `this` or `super` in any expression; otherwise, a compile-time error occurs. For example, if the first constructor of `ColoredPoint` in the example above were changed to:

```

        ColoredPoint(int x, int y) {
            this(x, y, color);
        }
    }
}

```

then a compile-time error would occur, because an instance variable cannot be used within a superclass constructor invocation.

An invocation of the constructor of the direct superclass, whether it actually appears as an explicit constructor invocation statement or is provided automatically ([§8.6.7](#)), performs an additional implicit action after a normal return of control from the constructor: all instance variables that have initializers are initialized at that time, in the textual order in which they appear in the class declaration. An invocation of another constructor in the same class using the keyword `this` does not perform this additional implicit action.

[§12.5](#) describes the creation and initialization of new class instances.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.6 Constructor Overloading

Overloading of constructors is identical in behavior to overloading of methods. The overloading is resolved at compile time by each class instance creation expression (\$15.8).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.7 Default Constructor

If a class contains no constructor declarations, then a *default constructor* that takes no parameters is automatically provided:

- If the class being declared is the primordial class `Object`, then the default constructor has an empty body.
- Otherwise, the default constructor takes no parameters and simply invokes the superclass constructor with no arguments.

A compile-time error occurs if a default constructor is provided by the compiler but the superclass does not have a constructor that takes no arguments.

If the class is declared `public`, then the default constructor is implicitly given the access modifier `public` ([§6.6](#)); otherwise, the default constructor has the default access implied by no access modifier. Thus, the example:

```
public class Point {  
    int x, y;  
}
```

is equivalent to the declaration:

```
public class Point {  
    int x, y;  
    public Point() { super(); }  
}
```

where the default constructor is `public` because the class `Point` is `public`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


8.6.8 Preventing Instantiation of a Class

A class can be designed to prevent code outside the class declaration from creating instances of the class by declaring at least one constructor, to prevent the creation of an implicit constructor, and declaring all constructors to be `private`. A `public` class can likewise prevent the creation of instances outside its package by declaring at least one constructor, to prevent creation of a default constructor with `public` access, and declaring no constructor that is `public`.

Thus, in the example:

```
class ClassOnly {
    private ClassOnly() { }
    static String just = "only the lonely";
}
```

the class `ClassOnly` cannot be instantiated, while in the example:

```
package just;
```

```
public class PackageOnly {
    PackageOnly() { }
    String[] justDesserts = { "cheesecake", "ice cream" };
}
```

the class `PackageOnly` can be instantiated only within the package `just`, in which it is declared.

Bow, bow, ye lower middle classes!
Bow, bow, ye tradesmen, bow, ye masses!
Blow the trumpets, bang the brasses!
Tantantara! Tzing! Boom!
--W.S. Gilbert, *Iolanthe*

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Interfaces

*My apple trees will never get across
And eat the cones under his pines, I tell him.
He only says "Good Fences Make Good Neighbors."
--Robert Frost, Mending Wall (1914)*

An interface declaration introduces a new reference type whose members are constants and abstract methods. This type has no implementation, but otherwise unrelated classes can implement it by providing implementations for its abstract methods.

Java programs can use interfaces to make it unnecessary for related classes to share a common abstract superclass or to add methods to `Object`.

An interface may be declared to be a *direct extension* of one or more other interfaces, meaning that it implicitly specifies all the abstract methods and constants of the interfaces it extends, except for any constants that it may hide.

A class may be declared to *directly implement* one or more interfaces, meaning that any instance of the class implements all the abstract methods specified by the interface or interfaces. A class necessarily implements all the interfaces that its direct superclasses and direct superinterfaces do. This (multiple) interface inheritance allows objects to support (multiple) common behaviors without sharing any implementation.

A variable whose declared type is an interface type may have as its value a reference to any object that is an instance of a class declared to implement the specified interface. It is not sufficient that the class happen to implement all the abstract methods of the interface; the class or one of its superclasses must actually be declared to implement the interface, or else the class is not considered to implement the interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.1 Interface Declarations

An interface declaration specifies a new reference type:

InterfaceDeclaration:

InterfaceModifiersopt interface Identifier

ExtendsInterfacesopt InterfaceBody

A compile-time error occurs if the *Identifier* naming an interface appears as the name of any other class or interface in the same package. A compile-time error also occurs if the *Identifier* naming an interface appears as the name by which a class or interface is to be known via a single-type-import declaration ([§7.5.1](#)) in the compilation unit containing the interface declaration. In the example:

```
class Point { int x, y; }
```

```
interface Point { void move(int dx, int dy); }
```

a compile-time error occurs because a `class` and an `interface` in the same package cannot have the same name.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.1.1 Scope of an Interface Type Name

The *Identifier* specifies the name of the interface and has as its scope the entire package in which it is declared. This is the same scoping rule as for class type names; see [§8.1.1](#) for an example involving classes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.1.2 Interface Modifiers

An interface declaration may be preceded by *interface modifiers*:

InterfaceModifiers:

InterfaceModifier

InterfaceModifiers InterfaceModifier

InterfaceModifier: one of

public abstract

The access modifier `public` is discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in an interface declaration.

9.1.2.1 abstract Interfaces

Every interface is implicitly `abstract`. This modifier is obsolete and should not be used in new Java programs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

9.1.3 Superinterfaces

If an `extends` clause is provided, then the interface being declared extends each of the other named interfaces and therefore inherits the methods and constants of each of the other named interfaces. These other named interfaces are the *direct superinterfaces* of the interface being declared. Any class that `implements` the declared interface is also considered to implement all the interfaces that this interface `extends` and that are accessible to the class.

ExtendsInterfaces:

extends InterfaceType

ExtendsInterfaces , InterfaceType

The following is repeated from [§4.3](#) to make the presentation here clearer:

InterfaceType:

TypeName

Each *InterfaceType* in the `extends` clause of an interface declaration must name an accessible interface type; otherwise a compile-time error occurs.

A compile-time error occurs if there is a circularity such that an interface directly or indirectly extends itself.

There is no analogue of the class `Object` for interfaces; that is, while every class is an extension of class `Object`, there is no single interface of which all interfaces are extensions.

The *superinterface* relationship is the transitive closure of the direct superinterface relationship. An interface *K* is a superinterface of interface *I* if either of the following is true:

- *K* is a direct superinterface of *I*.
- There exists an interface *J* such that *K* is a superinterface of *J*, and *J* is a superinterface of *I*, applying this definition recursively.

Interface *I* is said to be a *subinterface* of interface *K* whenever *K* is a superinterface of *I*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

9.1.4 Interface Body and Member Declarations

The body of an interface may declare members of the interface:

InterfaceBody:

{ InterfaceMemberDeclarationsopt }

InterfaceMemberDeclarations:

InterfaceMemberDeclaration

InterfaceMemberDeclarations InterfaceMemberDeclaration

InterfaceMemberDeclaration:

ConstantDeclaration

AbstractMethodDeclaration

The scope of the name of a member declared in an interface type is the entire body of the interface type declaration.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

9.1.5 Access to Interface Member Names

All interface members are implicitly `public`. They are accessible outside the package where the interface is declared if the interface is also declared `public` and the package containing the interface is accessible as described in [§7.1](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.2 Interface Members

The members of an interface are those members inherited from direct superinterfaces and those members declared in the interface.

The interface inherits, from the interfaces it extends, all members of those interfaces, except for fields that it hides and methods that it overrides.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.3 Field (Constant) Declarations

*The materials of action are variable,
but the use we make of them should be constant.*

--Epictetus (circa 60 A.D.), translated by Thomas Wentworth Higginson

ConstantDeclaration:

ConstantModifiers Type VariableDeclarator

ConstantModifiers: one of

public static final

Every field declaration in the body of an interface is implicitly `public`, `static`, and `final`. It is permitted, but strongly discouraged as a matter of style, to redundantly specify any or all of these modifiers for such fields.

A constant declaration in an interface must not include any of the modifiers `synchronized`, `transient`, or `volatile`, or a compile-time error occurs.

It is possible for an interface to inherit more than one field with the same name ([§8.3.3.3](#)). Such a situation does not in itself cause a compile-time error. However, any attempt within the body of the interface to refer to either field by its simple name will result in a compile-time error, because such a reference is ambiguous.

There might be several paths by which the same field declaration might be inherited from an interface. In such a situation, the field is considered to be inherited only once, and it may be referred to by its simple name without ambiguity.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.3.1 Initialization of Fields in Interfaces

Every field in the body of an interface must have an initialization expression, which need not be a constant expression. The variable initializer is evaluated and the assignment performed exactly once, when the interface is initialized ([§12.4](#)).

A compile-time error occurs if an initialization expression for an interface field contains a reference by simple name to the same field or to another field whose declaration occurs textually later in the same interface. Thus:

```
interface Test {  
    float f = j;  
    int j = 1;  
    int k = k+1;  
}
```

causes two compile-time errors, because `j` is referred to in the initialization of `f` before `j` is declared and because the initialization of `k` refers to `k` itself.

(One subtlety here is that, at run time, fields that are initialized with compile-time constant values are initialized first. This applies also to `static final` fields in classes ([§8.3.2.1](#)). This means, in particular, that these fields will never be observed to have their default initial values ([§4.5.4](#)), even by devious programs. See [§12.4.2](#) and [§13.4.8](#) for more discussion.)

If the keyword `this` ([§15.7.2](#)) or the keyword `super` ([15.10.2](#), [15.11](#)) occurs in an initialization expression for a field of an interface, then a compile-time error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.3.2 Examples of Field Declarations

The following example illustrates some (possibly subtle) points about field declarations.

9.3.2.1 Ambiguous Inherited Fields

If two fields with the same name are inherited by an interface because, for example, two of its direct superinterfaces declare fields with that name, then a single *ambiguous member* results. Any use of this ambiguous member will result in a compile-time error. Thus in the example:

```
interface BaseColors {
    int RED = 1, GREEN = 2, BLUE = 4;
}

interface RainbowColors extends BaseColors {
    int YELLOW = 3, ORANGE = 5, INDIGO = 6, VIOLET = 7;
}

interface PrintColors extends BaseColors {
    int YELLOW = 8, CYAN = 16, MAGENTA = 32;
}

interface LotsOfColors extends RainbowColors, PrintColors {
    int FUCHSIA = 17, VERMILION = 43, CHARTREUSE = RED+90;
}
```

the interface `LotsOfColors` inherits two fields named `YELLOW`. This is all right as long as the interface does not contain any reference by simple name to the field `YELLOW`. (Such a reference could occur within a variable initializer for a field.)

Even if interface `PrintColors` were to give the value 3 to `YELLOW` rather than the value 8, a reference to field `YELLOW` within interface `LotsOfColors` would still be considered ambiguous.

9.3.2.2 Multiply Inherited Fields

If a single field is inherited multiple times from the same interface because, for example, both this interface and one of this interface's direct superinterfaces extend the interface that declares the field, then only a single member results. This situation does not in itself cause a compile-time error.

In the example in the previous section, the fields `RED`, `GREEN`, and `BLUE` are inherited by interface `LotsOfColors` in more than one way, through interface `RainbowColors` and also through interface `PrintColors`, but the reference to field `RED` in interface `LotsOfColors` is not considered ambiguous because only one actual declaration of the field `RED` is involved.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

9.4 Abstract Method Declarations

AbstractMethodDeclaration:

AbstractMethodModifiersopt ResultType MethodDeclarator Throwsopt ;

AbstractMethodModifiers:

AbstractMethodModifier

AbstractMethodModifiers AbstractMethodModifier

AbstractMethodModifier: one of

public abstract

The access modifier `public` is discussed in [§6.6](#). A compile-time error occurs if the same modifier appears more than once in an abstract method declaration.

Every method declaration in the body of an interface is implicitly `abstract`, so its body is always represented by a semicolon, not a block. For compatibility with older versions of Java, it is permitted but discouraged, as a matter of style, to redundantly specify the `abstract` modifier for methods declared in interfaces.

Every method declaration in the body of an interface is implicitly `public`. It is permitted, but strongly discouraged as a matter of style, to redundantly specify the `public` modifier for interface methods.

Note that a method declared in an interface must not be declared `static`, or a compile-time error occurs, because in Java `static` methods cannot be `abstract`.

Note that a method declared in an interface must not be declared `native` or `synchronized`, or a compile-time error occurs, because those keywords describe implementation properties rather than interface properties. However, a method declared in an interface may be implemented by a method that is declared `native` or `synchronized` in a class that implements the interface.

Note that a method declared in an interface must not be declared `final` or a compile-time error occurs. However, a method declared in an interface may be implemented by a method that is declared `final` in a class that implements the interface.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

9.4.1 Inheritance and Overriding

If the interface declares a method, then the declaration of that method is said to *override* any and all methods with the same signature in the superinterfaces of the interface that would otherwise be accessible to code in this interface.

If a method declaration in an interface overrides the declaration of a method in another interface, a compile-time error occurs if the methods have different return types or if one has a return type and the other is `void`. Moreover, a method declaration must not have a `throws` clause that conflicts ([§8.4.4](#)) with that of any method that it overrides; otherwise, a compile-time error occurs.

Methods are overridden on a signature-by-signature basis. If, for example, an interface declares two `public` methods with the same name, and a subinterface overrides one of them, the subinterface still inherits the other method.

An interface inherits from its direct superinterfaces all methods of the superinterfaces that are not overridden by a declaration in the interface.

It is possible for an interface to inherit more than one method with the same signature ([§8.4.2](#)). Such a situation does not in itself cause a compile-time error. The interface is considered to inherit all the methods. However, a compile-time error occurs if, for any two such inherited methods, either they have different return types or one has a return type and the other is `void`. (The `throws` clauses do not cause errors in this case.)

There might be several paths by which the same method declaration is inherited from an interface. This fact causes no difficulty and never of itself results in a compile-time error.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.4.2 Overloading

If two methods of an interface (whether both declared in the same interface, or both inherited by a interface, or one declared and one inherited) have the same name but different signatures, then the method name is said to be *overloaded*. This fact causes no difficulty and never of itself results in a compile-time error. There is no required relationship between the return types or between the `throws` clauses of two methods with the same name but different signatures.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


9.4.3 Examples of Abstract Method Declarations

The following examples illustrate some (possibly subtle) points about abstract method declarations.

9.4.3.1 Example: Overriding

Methods declared in interfaces are `abstract` and thus contain no implementation. About all that can be accomplished by an overriding method declaration, other than to affirm a method signature, is to restrict the exceptions that might be thrown by an implementation of the method. Here is a variation of the example shown in [§8.4.3.1](#):

```
class BufferEmpty extends Exception {
    BufferEmpty() { super(); }
    BufferEmpty(String s) { super(s); }
}

class BufferError extends Exception {
    BufferError() { super(); }
    BufferError(String s) { super(s); }
}

public interface Buffer {
    char get() throws BufferEmpty, BufferError;
}

public interface InfiniteBuffer extends Buffer {
    char get() throws BufferError;
    // override
}
```

9.4.3.2 Example: Overloading

In the example code:

```
interface PointInterface {
    void move(int dx, int dy);
}

interface RealPointInterface extends PointInterface {
    void move(float dx, float dy);
    void move(double dx, double dy);
}
```


the method name `move` is overloaded in interface `RealPointInterface` with three different signatures, two of them declared and one inherited. Any class that implements interface `RealPointInterface` must provide implementations of all three method signatures.

*Death, life, and sleep, reality and thought,
Assist me, God, their boundaries to know . . .
--William Wordsworth, *Maternal Grief**

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Arrays

Even Solomon in all his glory was not arrayed like one of these.

--Matthew 6:29

Java *arrays* are objects (§4.3.1), are dynamically created, and may be assigned to variables of type `Object` (§4.3.2). All methods of class `Object` may be invoked on an array.

An array object contains a number of variables. The number of variables may be zero, in which case the array is said to be *empty*. The variables contained in an array have no names; instead they are referenced by array access expressions that use nonnegative integer index values. These variables are called the *components* of the array. If an array has n components, we say n is the *length* of the array; the components of the array are referenced using integer indices from 0 to $\{ewc msdn cd, EWGraphic, LNG0n 0 /a "langref0ANC.BMP"\}$, inclusive.

All the components of an array have the same type, called the *component type* of the array. If the component type of an array is T , then the type of the array itself is written $T[]$.

The component type of an array may itself be an array type. The components of such an array may contain references to subarrays. If, starting from any array type, one considers its component type, and then (if that is also an array type) the component type of that type, and so on, eventually one must reach a component type that is not an array type; this is called the *element type* of the original array, and the components at this level of the data structure are called the *elements* of the original array.

There is one situation in which an element of an array can be an array: if the element type is `Object`, then some or all of the elements may be arrays, because any array object can be assigned to any variable of type `Object`.

{ewl msdn cd.dll, ewcright, /c"Microsoft"}

10.1 Array Types

An array type is written as the name of an element type followed by some number of empty pairs of square brackets `[]`. The number of bracket pairs indicates the depth of array nesting. An array's length is not part of its type.

The element type of an array may be any type, whether primitive or reference. In particular:

- Arrays with an interface type as the component type are allowed. The elements of such an array may have as their value a null reference or instances of any class type that implements the interface.
- Arrays with an `abstract` class type as the component type are allowed. The elements of such an array may have as their value a null reference or instances of any subclass of the `abstract` class that is not itself `abstract`.

Array types are used in declarations and in cast expressions (§15.15).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.2 Array Variables

A variable of array type holds a reference to an object. Declaring a variable of array type does not create an array object or allocate any space for array components. It creates only the variable itself, which can contain a reference to an array. However, the initializer part of a declarator (§8.3) may create an array, a reference to which then becomes the initial value of the variable.

Because an array's length is not part of its type, a single variable of array type may contain references to arrays of different lengths.

Here are examples of declarations of array variables that do not create arrays:

```
int[] ai;                // array of int
short[][] as;            // array of array of
short
Object[] ao,             // array of Object
        otherAo;         // array of Object
short s,                 // scalar short
aas[][];                // array of array of short
```

Here are some examples of declarations of array variables that create array objects:

```
Exception ae[] = new Exception[3];
Object aao[][] = new Exception[2][3];
int[] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };
char ac[] = { 'n', 'o', 't', ' ', 'a', ' ',
              'S', 't', 'r', 'i', 'n', 'g' };
String[] aas = { "array", "of", "String", };
```

The [] may appear as part of the type at the beginning of the declaration, or as part of the declarator for a particular variable, or both, as in this example:

```
byte[] rowvector, colvector, matrix[];
```

This declaration is equivalent to:

```
byte rowvector[], colvector[], matrix[][];
```

Once an array object is created, its length never changes. To make an array variable refer to an array of different length, a reference to a different array must be assigned to the variable.

If an array variable *v* has type *A*[], where *A* is a reference type, then *v* can hold a reference to an instance of any array type *B*[], provided *B* can be assigned to *A*. This may result in a run-time exception on a later assignment; see §10.10 for a discussion.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.3 Array Creation

An array is created by an array creation expression ([§15.9](#)) or an array initializer ([§10.6](#)).

An array creation expression specifies the element type, the number of levels of nested arrays, and the length of the array for at least one of the levels of nesting. The array's length is available as a final instance variable `length`.

An array initializer creates an array and provides initial values for all its components. (Contrast this with C and C++, where it is possible for an array initializer to specify initial values for some but not all of the components of an array.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.4 Array Access

A component of an array is accessed by an array access expression (§15.12) that consists of an expression whose value is an array reference followed by an indexing expression enclosed by `[` and `]`, as in `A[i]`. All arrays are 0-origin. An array with length n can be indexed by the integers 0 to $n-1$.

Arrays must be indexed by `int` values; `short`, `byte`, or `char` values may also be used as index values because they are subjected to unary numeric promotion (§5.6.1) and become `int` values. An attempt to access an array component with a `long` index value results in a compile-time error.

All array accesses are checked at run time; an attempt to use an index that is less than zero or greater than or equal to the length of the array causes an `IndexOutOfBoundsException` to be thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.5 A Simple Example

The example:

```
class Gauss {
    public static void main(String[] args) {
        int[] ia = new int[101];
        for (int i = 0; i < ia.length; i++)
            ia[i] = i;
        int sum = 0;
        for (int i = 0; i < ia.length; i++)
            sum += ia[i];
        System.out.println(sum);
    }
}
```

that produces output:

5050

declares a variable `ia` that has type array of `int`, that is, `int[]`. The variable `ia` is initialized to reference a newly created array object, created by an array creation expression (§15.9). The array creation expression specifies that the array should have 101 components. The length of the array is available using the field `length`, as shown.

The example program fills the array with the integers from 0 to 100, sums these integers, and prints the result.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

10.6 Arrays Initializers

An *array initializer* may be specified in a declaration, creating an array and providing some initial values:

ArrayInitializer:

{ VariableInitializersopt ,opt }

VariableInitializers:

VariableInitializer

VariableInitializers , VariableInitializer

The following is repeated from §8.3 to make the presentation here clearer:

VariableInitializer:

Expression

ArrayInitializer

An array initializer is written as a comma-separated list of expressions, enclosed by braces "{" and "}".

The length of the constructed array will equal the number of expressions.

Each expression specifies a value for one array component. Each expression must be assignment-compatible (§5.2) with the array's component type, or a compile-time error results.

If the component type is itself an array type, then the expression specifying a component may itself be an array initializer; that is, array initializers may be nested.

A trailing comma may appear after the last expression in an array initializer and is ignored.

As an example:

```
class Test {
    public static void main(String[] args) {
        int ia[][] = { {1, 2}, null };
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                System.out.println(ia[i][j]);
    }
}
```

prints:

1
2

before causing a `NullPointerException` in trying to index the second component of the array `ia`, which is a null reference.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.7 Array Members

The members of an array type are all of the following:

- The public final field `length`, which contains the number of components of the array (`length` may be positive or zero)
- The public method `clone`, which overrides the method of the same name in class `Object` and throws no checked exceptions
- All the members inherited from class `Object`; the only method of `Object` that is not inherited is its `clone` method

An array thus has the same methods as the following class:

```
class A implements Cloneable {
    public final int length = X;
    public Object clone() {
        try {
            return super.clone();
        } catch (CloneNotSupportedException e) {
            throw new InternalError(e.getMessage());
        }
    }
}
```

Every array implements interface `Cloneable`. That arrays are cloneable is shown by the test program:

```
class Test {
    public static void main(String[] args) {
        int ia1[] = { 1, 2 };
        int ia2[] = (int[])ia1.clone();
        System.out.print((ia1 == ia2) + " ");
        ia1[1]++;
        System.out.println(ia2[1]);
    }
}
```

which prints:

```
false 2
```

showing that the components of the arrays referenced by `ia1` and `ia2` are different variables. (In some early implementations of Java this example failed to compile because the compiler incorrectly believed that the clone method for an array could throw a `CloneNotSupportedException`.)

A `clone` of a multidimensional array is shallow, which is to say that it creates only a single new array. Subarrays are shared, as shown by the example program:


```

class Test {
    public static void main(String[] args) throws Throwable {
        int ia[][] = { { 1 , 2}, null };
        int ja[][] = (int[][])ia.clone();
        System.out.print((ia == ja) + " ");
        System.out.println(ia[0] == ja[0] && ia[1] == ja[1]);
    }
}

```

which prints:

```
false true
```

showing that the `int[]` array that is `ia[0]` and the `int[]` array that is `ja[0]` are the same array.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.8 Class Objects for Arrays

Every array has an associated `Class` object, shared with all other arrays with the same component type. The superclass of an array type is considered to be `Object`, as shown by the following example code:

```
class Test {
    public static void main(String[] args) {
        int[] ia = new int[3];
        System.out.println(ia.getClass());
        System.out.println(ia.getClass().getSuperclass());
    }
}
```

which prints:

```
class [I
class java.lang.Object
```

where the string "[I" is the run-time type signature for the class object "array with component type int" ([§20.1.1](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

10.9 An Array of Characters is Not a String

In Java, unlike C, an array of `char` is not a `String` ([§20.12](#)), and neither a `String` nor an array of `char` is terminated by `'\u0000'` (the NUL character).

A Java `String` object is immutable, that is, its contents never change, while an array of `char` has mutable elements. The method `toCharArray` in class `String` returns an array of characters containing the same character sequence as a `String`. The class `StringBuffer` implements useful methods on mutable arrays of characters ([§20.13](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


10.10 Array Store Exception

If an array variable *v* has type *A*[], where *A* is a reference type, then *v* can hold a reference to an instance of any array type *B*[], provided *B* can be assigned to *A*.

Thus, the example:

```
class Point { int x, y; }

class ColoredPoint extends Point { int color; }

class Test {
    public static void main(String[] args) {
        ColoredPoint[] cpa = new ColoredPoint[10];
        Point[] pa = cpa;
        System.out.println(pa[1] == null);
        try {
            pa[0] = new Point();
        } catch (ArrayStoreException e) {
            System.out.println(e);
        }
    }
}
```

produces the output:

```
true
java.lang.ArrayStoreException
```

Here the variable *pa* has type *Point*[] and the variable *cpa* has as its value a reference to an object of type *ColoredPoint*[], A *ColoredPoint* can be assigned to a *Point*; therefore, the value of *cpa* can be assigned to *pa*.

A reference to this array *pa*, for example, testing whether *pa*[1] is *null*, will not result in a run-time type error. This is because the element of the array of type *ColoredPoint*[] is a *ColoredPoint*, and every *ColoredPoint* can stand in for a *Point*, since *Point* is the superclass of *ColoredPoint*.

On the other hand, an assignment to the array *pa* can result in a run-time error. At compile time, an assignment to an element of *pa* is checked to make sure that the value assigned is a *Point*. But since *pa* holds a reference to an array of *ColoredPoint*, the assignment is valid only if the type of the value assigned at run-time is, more specifically, a *ColoredPoint*.

Java checks for such a situation at run-time to ensure that the assignment is valid; if not, an *ArrayStoreException* is thrown. More formally: an assignment to an element of an array whose type is *A*[], where *A* is a reference type, is checked at run-time to ensure that the value assigned can be assigned to the actual element type of the array, where the actual element type may be any

reference type that is assignable to A.

*At length burst in the argent revelry,
With plume, tiara, and all rich array . . .*
--John Keats, *The Eve of St. Agnes* (1819)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Exceptions

If anything can go wrong, it will.

--Finagle's Law (often incorrectly attributed to Murphy, whose law is rather different--which only goes to show that Finagle was right)

When a Java program violates the semantic constraints of the Java language, a Java Virtual Machine signals this error to the program as an *exception*. An example of such a violation is an attempt to index outside the bounds of an array. Some programming languages and their implementations react to such errors by peremptorily terminating the program; other programming languages allow an implementation to react in an arbitrary or unpredictable way. Neither of these approaches is compatible with the design goals of Java: to provide portability and robustness. Instead, Java specifies that an exception will be thrown when semantic constraints are violated and will cause a non-local transfer of control from the point where the exception occurred to a point that can be specified by the programmer. An exception is said to be *thrown* from the point where it occurred and is said to be *caught* at the point to which control is transferred.

Java programs can also throw exceptions explicitly, using `throw` statements (§14.16). This provides an alternative to the old-fashioned style of handling error conditions by returning funny values, such as the integer value `-1` where a negative value would not normally be expected. Experience shows that too often such funny values are ignored or not checked for by callers, leading to programs that are not robust, exhibit undesirable behavior, or both.

Every exception is represented by an instance of the class `Throwable` or one of its subclasses; such an object can be used to carry information from the point at which an exception occurs to the handler that catches it. Handlers are established by `catch` clauses of `try` statements (§14.18). During the process of throwing an exception, a Java Virtual Machine abruptly completes, one by one, any expressions, statements, method and constructor invocations, static initializers, and field initialization expressions that have begun but not completed execution in the current thread. This process continues until a handler is found that indicates that it handles that particular exception by naming the class of the exception or a superclass of the class of the exception. If no such handler is found, then the method `uncaughtException` (§20.21.31) is invoked for the `ThreadGroup` that is the parent of the current thread--thus every effort is made to avoid letting an exception go unhandled.

The Java exception mechanism is integrated with the Java synchronization model (§17), so that locks are released as `synchronized` statements (§14.17) and invocations of `synchronized` methods (§8.4.3.5, §15.11) complete abruptly.

This chapter describes the different causes of exceptions (§11.1). It details how exceptions are checked at compile time (§11.2) and processed at run time (§11.3). A detailed example (§11.4) is then followed by an explanation of the exception hierarchy and the standard exception classes (§11.5).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.1 The Causes of Exceptions

If we do not succeed, then we run the risk of failure.

--J. Danforth Quayle (1990)

An exception is thrown for one of three *reasons*:

- An abnormal execution condition was synchronously detected by a Java Virtual Machine. Such conditions arise because:
 - evaluation of an expression violates the normal semantics of the Java language, such as an integer divide by zero, as summarized in §15.5
 - an error occurs in loading or linking part of the Java program (§12.2, §12.3)
 - some limitation a resource is exceeded, such as using too much memory

These exceptions are not thrown at an arbitrary point in the program, but rather at a point where they are specified as a possible result of an expression evaluation or statement execution.

- A `throw` statement (§14.16) was executed in Java code.
- An asynchronous exception occurred either because:
 - the method `stop` of class `Thread` (§20.20.16) was invoked
 - an internal error has occurred in the virtual machine (§11.5.2.2)

Exceptions are represented by instances of the class `Throwable` and instances of its subclasses. These classes are, collectively, the *exception classes*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.2 Compile-Time Checking of Exceptions

The Java language checks, at compile time, that a Java program contains handlers for *checked exceptions*, by analyzing which checked exceptions can result from execution of a method or constructor. For each checked exception which is a possible result, the `throws` clause for the method (§8.4.4) or constructor (§8.6.4) must mention the class of that exception or one of the superclasses of the class of that exception. This compile-time checking for the presence of exception handlers is designed to reduce the number of exceptions which are not properly handled.

The *unchecked exceptions classes* are the class `RuntimeException` and its subclasses, and the class `Error` and its subclasses. All other exception classes are *checked exception classes*. The standard Java API defines a number of exception classes, both checked and unchecked. Additional exception classes, both checked and unchecked, may be declared by Java programmers. See §11.5 for a description of the Java exception class hierarchy and the exception classes defined by the standard Java API and Java Virtual Machine.

The checked exception classes named in the `throws` clause are part of the contract between the implementor and user of the method or constructor. The `throws` clause of an overriding method may not specify that this method will result in throwing any checked exception which the overridden method is not permitted, by its `throws` clause, to throw. When interfaces are involved, more than one method declaration may be overridden by a single overriding declaration. In this case, the overriding declaration must have a `throws` clause that is compatible with *all* the overridden declarations (§9.4).

Variable initializers for fields (§8.3.2) and static initializers (§8.5) must not result in a checked exception; if one does, a compile-time error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

11.2.1 Why Errors are Not Checked

Those unchecked exception classes which are the *error classes* (`Error` and its subclasses) are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A Java program declaring such exceptions would be cluttered, pointlessly.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.2.2 Why Runtime Exceptions are Not Checked

The *runtime exception classes* (`RuntimeException` and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of Java, having to declare such exceptions would not aid significantly in establishing the correctness of Java programs. Many of the operations and constructs of the Java language can result in runtime exceptions. The information available to a Java compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such runtime exceptions cannot occur, even though this may be obvious to the Java programmer. Requiring such exception classes to be declared would simply be an irritation to Java programmers.

For example, certain code might implement a circular data structure that, by construction, can never involve `null` references; the programmer can then be certain that a `NullPointerException` cannot occur, but it would be difficult for a compiler to prove it. The theorem-proving technology that is needed to establish such global properties of data structures is beyond the scope of this Java Language Specification.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.3 Handling of an Exception

When an exception is thrown, control is transferred from the code that caused the exception to the nearest dynamically-enclosing `catch` clause of a `try` statement (§14.18) that handles the exception.

A statement or expression is *dynamically enclosed* by a `catch` clause if it appears within the `try` block of the `try` statement of which the `catch` clause is a part, or if the caller of the statement or expression is dynamically enclosed by the `catch` clause.

The *caller* of a statement or expression depends on where it occurs:

- If within a method, then the caller is the method invocation expression (§15.11) that was executed to cause the method to be invoked.
- If within a constructor or the initializer for an instance variable, then the caller is the class instance creation expression (§15.8) or the method invocation of `newInstance` that was executed to cause an object to be created.
- If within a static initializer or an initializer for a `static` variable, then the caller is the expression that used the class or interface so as to cause it to be initialized.

Whether a particular `catch` clause *handles* an exception is determined by comparing the class of the object that was thrown to the declared type of the parameter of the `catch` clause. The `catch` clause handles the exception if the type of its parameter is the class of the exception or a superclass of the class of the exception. Equivalently, a `catch` clause will catch any exception object that is an `instanceof` (§15.19.2) the declared parameter type.

The control transfer that occurs when an exception is thrown causes abrupt completion of expressions (§15.5) and statements (§14.1) until a `catch` clause is encountered that can handle the exception; execution then continues by executing the block of that `catch` clause. The code that caused the exception is never resumed.

If no `catch` clause handling an exception can be found, then the current thread (the thread that encountered the exception) is terminated, but only after all `finally` clauses have been executed and the method `uncaughtException` (§20.21.31) has been invoked for the `ThreadGroup` that is the parent of the current thread.

In situations where it is desirable to ensure that one block of code is always executed after another, even if that other block of code completes abruptly, a `try` statement with a `finally` clause (§14.18.2) may be used. If a `try` or `catch` block in a `try-finally` or `try-catch-finally` statement completes abruptly, then the `finally` clause is executed during propagation of the exception, even if no matching `catch` clause is ultimately found. If a `finally` clause is executed because of abrupt completion of a `try` block and the `finally` clause itself completes abruptly, then the reason for the abrupt completion of the `try` block is discarded and the new reason for abrupt completion is propagated from there.

The exact rules for abrupt completion and for the catching of exceptions are specified in detail with the specification of each statement in §14 and for expressions in §15 (especially §15.5).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.3.1 Exceptions are Precise

Exceptions in Java are *precise*: when the transfer of control takes place, all effects of the statements executed and expressions evaluated before the point from which the exception is thrown must appear to have taken place. No expressions, statements, or parts thereof that occur after the point from which the exception is thrown may appear to have been evaluated. If optimized code has speculatively executed some of the expressions or statements which follow the point at which the exception occurs, such code must be prepared to hide this speculative execution from the user-visible state of the Java program.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.3.2 Handling Asynchronous Exceptions

Most exceptions in Java occur synchronously as a result of an action by the thread in which they occur, and at a point in the Java program that is specified to possibly result in such an exception. An asynchronous exception is, by contrast, an exception that can potentially occur at any point in the execution of a Java program.

Asynchronous exceptions are rare in Java. They occur only as a result of:

- An invocation of the `stop` methods of class `Thread` ([§20.20.15](#), [§20.20.16](#)) or `ThreadGroup` ([§20.21.8](#), [§20.21.9](#))
- An `InternalError` ([§11.5.2.2](#)) in the Java Virtual Machine

The `stop` methods may be invoked by one thread to affect another thread or all the threads in a specified thread group. They are asynchronous because they may occur at any point in the execution of the other thread or threads. An `InternalError` is considered asynchronous so that it may be handled using the same mechanism that handles the `stop` method, as will now be described.

Java permits a small but bounded amount of execution to occur before an asynchronous exception is thrown. This delay is permitted to allow optimized code to detect and throw these exceptions at points where it is practical to handle them while obeying the semantics of the Java language.

A simple implementation might poll for asynchronous exceptions at the point of each control transfer instruction. Since a Java program has a finite size, this provides a bound on the total delay in detecting an asynchronous exception. Since no asynchronous exception will occur between control transfers, the code generator has some flexibility to reorder computation between control transfers for greater performance.

The paper *Polling Efficiently on Stock Hardware* by Mark Feeley, *Proc. 1993 Conference on Functional Programming and Computer Architecture*, Copenhagen, Denmark, pp. 179-187, is recommended as further reading.

Like all exceptions, asynchronous exceptions are precise ([§11.3.1](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

11.4 An Example of Exceptions

Consider the following example:

```
class TestException extends Exception {

    TestException() { super(); }

    TestException(String s) { super(s); }

}

class Test {

    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {

            try {
                thrower(args[i]);
                System.out.println("Test \"" + args[i] +
                    "\" didn't throw an exception");
            } catch (Exception e) {
                System.out.println("Test \"" + args[i] +
                    "\" threw a " + e.getClass() +
                    "\"\n          with message: " + e.getMessage());
            }

        }

    }

    static int thrower(String s) throws TestException {
        try {
            if (s.equals("divide")) {
                int i = 0;
                return i/i;
            }
            if (s.equals("null")) {
                s = null;
                return s.length();
            }
            if (s.equals("test"))
                throw new TestException("Test message");
            return 0;
        } finally {
            System.out.println("[thrower(\"" + s +
                "\") done]");
        }

    }

}
```


If we execute the test program, passing it the arguments:

```
divide null not test
```

it produces the output:

```
[thrower("divide") done]
Test "divide" threw a class java.lang.ArithmeticException
    with message: / by zero
[thrower("null") done]
Test "null" threw a class java.lang.NullPointerException
    with message: null
[thrower("not") done]
Test "not" didn't throw an exception
[thrower("test") done]
Test "test" threw a class TestException
    with message: Test message
```

This example declares an exception class `TestException`. The `main` method of class `Test` invokes the `thrower` method four times, causing exceptions to be thrown three of the four times. The `try` statement in method `main` catches each exception that the `thrower` throws. Whether the invocation of `thrower` completes normally or abruptly, a message is printed describing what happened.

The declaration of the method `thrower` must have a `throws` clause because it can throw instances of `TestException`, which is a checked exception class ([§11.2](#)). A compile-time error would occur if the `throws` clause were omitted.

Notice that the `finally` clause is executed on every invocation of `thrower`, whether or not an exception occurs, as shown by the `"[thrower(...) done]"` output that occurs for each invocation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.5 The Exception Hierarchy

The possible exceptions in a Java program are organized in a hierarchy of classes, rooted at class `Throwable` (§11.5, §20.22), a direct subclass of `Object`. The classes `Exception` and `Error` are direct subclasses of `Throwable`. The class `RuntimeException` is a direct subclass of `Exception`.

The exception classes declared by the standard packages `java.lang`, `java.util`, `java.io` and `java.net` are called the *standard exception classes*.

Java programs can use the pre-existing exception classes in `throw` statements, or define additional exception classes, as subclasses of `Throwable` or of any of its subclasses, as appropriate. To take advantage of Java's compile-time checking for exception handlers, it is typical to define most new exception classes as checked exception classes, specifically as subclasses of `Exception` that are not subclasses of `RuntimeException`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.5.1 The Classes Exception and RuntimeException

The class `Exception` is the superclass of all the exceptions that ordinary programs may wish to recover from.

11.5.1.1 Standard Runtime Exceptions

The class `RuntimeException` is a subclass of class `Exception`. The subclasses of `RuntimeException` are unchecked exception classes.

Package `java.lang` defines the following standard unchecked runtime exceptions, which, like all other classes in package `java.lang`, are implicitly imported and therefore may be referred to by their simple names:

- `ArithmeticException`: An exceptional arithmetic situation has arisen, such as an integer division (§15.16.2) operation with a zero divisor.
- `ArrayStoreException`: An attempt has been made to store into an array component a value whose class is not assignment compatible with the component type of the array (§10.10, §15.25.1).
- `ClassCastException`: An attempt has been made to cast (§5.4, §15.15) a reference to an object to an inappropriate type.
- `IllegalArgumentException`: A method was passed an invalid or inappropriate argument or invoked on an inappropriate object. Subclasses of this class include:
 - `IllegalThreadStateException`: A thread was not in an appropriate state for a requested operation.
 - `NumberFormatException`: An attempt was made to convert a `String` to a value of a numeric type, but the `String` did not have an appropriate format.
- `IllegalMonitorStateException`: A thread has attempted to wait on (§20.1.6, §20.1.7, §20.1.8) or notify (§20.1.9, §20.1.10) other threads waiting on an object that it has not locked.
- `IndexOutOfBoundsException`: Either an index of some sort (such as to an array, a string, or a vector) or a subrange, specified either by two index values or by an index and a length, was out of range.
- `NegativeArraySizeException`: An attempt was made to create an array with a negative length (§15.9).
- `NullPointerException`: An attempt was made to use a null reference in a case where an object reference was required.
- `SecurityException`: A security violation was detected (§20.17).

Package `java.util` defines the following additional standard unchecked runtime exceptions:

- `java.util.EmptyStackException`: An attempt was made to access an element of an empty stack.
- `java.util.NoSuchElementException`: An attempt was made to access an element of an empty vector.

11.5.1.2 Standard Checked Exceptions

The standard subclasses of `Exception` other than `RuntimeException` are all checked exception classes.

Package `java.lang` defines the following standard exceptions, which, like all other classes in package `java.lang`, are implicitly imported and therefore may be referred to by their simple names:

- `ClassNotFoundException`: A class or interface with a specified name could not be found ([§20.3.8](#)).
- `CloneNotSupportedException`: The `clone` method ([§20.1.5](#)) of class `Object` has been invoked to clone an object, but the class of that object does not implement the `Cloneable` interface.
- `IllegalAccessException`: An attempt has been made to load a class using a string giving its fully qualified name, but the currently executing method does not have access to the definition of the specified class because the class is not `public` and is in another package.
- `InstantiationException`: An attempt was made to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface, is `abstract`, or is an array.
- `InterruptedException`: The current thread was waiting, and another thread has interrupted the current thread, using the `interrupt` method of class `Thread` ([§20.20.31](#)).

Package `java.io` defines the following additional standard exceptions:

- `java.io.IOException`: A requested I/O operation could not be completed normally. Subclasses of this class include:
 - `java.io.EOFException`: End of file has been encountered before normal completion of an input operation.
 - `java.io.FileNotFoundException`: A file with the name specified by a file name string or path was not found within the file system.
 - `java.io.InterruptedIOException`: The current thread was waiting for completion of an I/O operation, and another thread has interrupted the current thread, using the `interrupt` method of class `Thread` ([§20.20.31](#)).
 - `java.io.UTFDataFormatException`: A requested conversion of a string to or from Java modified UTF-8 format could not be completed ([§22.1.15](#), [§22.2.14](#)) because the string was too long or because the purported UTF-8 data was not the result of encoding a Unicode string into UTF-8.

The standard package `java.net` defines the following additional subclasses of `java.io.IOException`:

u `java.net.MalformedURLException`: A string that was provided as a URL, or as part of a URL, had an inappropriate format or specified an unknown protocol.

- `java.net.ProtocolException`: Some aspect of a network protocol was not correctly carried out.
- `java.net.SocketException`: An operation involving a socket could not be completed normally.
- `java.net.UnknownHostException`: The name of a network host could not be resolved to a network address.
- `java.net.UnknownServiceException`: The network connection cannot support the requested service.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


11.5.2 The Class Error

The class `Error` and its standard subclasses are exceptions from which ordinary programs are not ordinarily expected to recover. The class `Error` is a separate subclass of `Throwable`, distinct from `Exception` in the class hierarchy, to allow programs to use the idiom:

```
} catch (Exception e) {
```

to catch all exceptions from which recovery may be possible without catching errors from which recovery is typically not possible.

Package `java.lang` defines all the error classes described here. These classes, like all other classes in package `java.lang`, are implicitly imported and therefore may be referred to by their simple names.

11.5.2.1 Loading and Linkage Errors

A Java Virtual Machine throws an object that is an instance of a subclass of `LinkageError` when a loading, linkage, preparation, verification or initialization error occurs:

- The loading process is described in [§12.2](#). The errors `ClassFormatError`, `ClassCircularityError`, and `NoClassDefFoundError` are described there.
- The linking process is described in [§12.3](#). The linking errors are described there. These errors include `IllegalAccessError`, `InstantiationError`, `NoSuchFieldError`, and `NoSuchMethodError`, all of which are subclasses of `IncompatibleClassChangeError`, and, also, `UnsatisfiedLinkError`.
- The class verification process is described in [§12.3.1](#). The verification failure error `VerifyError` is described there.
- The class preparation process is described in [§12.3.2](#). The preparation error described there is `AbstractMethodError`.
- The class initialization process is described in [§12.4](#). A virtual machine will throw the error `ExceptionInInitializerError` if execution of a static initializer or of an initializer for a static field results in an exception that is not an `Error` or a subclass of `Error`.

11.5.2.2 Virtual Machine Errors

A Java Virtual Machine throws an object that is an instance of a subclass of the class `VirtualMachineError` when an internal error or resource limitation prevents it from implementing the semantics of the Java Language. This language specification and the Java Virtual Machine Specification define the following virtual machine errors:

- **`InternalError`:** An internal error has occurred in a Java Virtual Machine, because of a fault in the software implementing the virtual machine, a fault in the underlying host system software, or a fault in the hardware. This error is delivered asynchronously when it is detected, and may occur at any point in a Java program.
- **`OutOfMemoryError`:** A Java Virtual Machine has run out of either virtual or physical memory, and the automatic storage manager wasn't able to reclaim enough memory to satisfy an object creation request.

- `StackOverflowError`: A Java Virtual Machine has run out of stack space for a thread, typically because the thread is doing an unbounded number of recursive invocations due to a fault in the executing program.
- `UnknownError`: An exception or error has occurred but, for some reason, a Java Virtual Machine is unable to report the actual exception or error.

A sophisticated Java program may be designed to handle `OutOfMemoryError` and attempt to recover from it, perhaps by carefully dropping references to objects.

We are exploring enhancements to Java to simplify handling of out-of-memory conditions. One possibility would be to support automatic suspension of a thread which encounters an `OutOfMemoryError` and allow another thread to handle the `error` situation. Such a technique might also permit a Java program to recover from a `StackOverflowError` if this overflow does not result from a nonterminating recursion. Suggestions for other approaches are welcomed.

No rule is so general, which admits not some exception.

--Robert Burton (1576-1640)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Execution

We must all hang together, or assuredly we shall all hang separately.

--Benjamin Franklin (July 4, 1776)

This chapter specifies activities that occur during execution of a Java program. It is organized around the life cycle of a Java Virtual Machine and of the classes, interfaces, and objects that form a Java program.

A Java Virtual Machine starts up by loading a specified class and then invoking the method `main` in this specified class. Section [§12.1](#) outlines the loading, linking, and initialization steps involved in executing `main`, as an introduction to the concepts in this chapter. Further sections specify the details of loading ([§12.2](#)), linking ([§12.3](#)), and initialization ([§12.4](#)).

The chapter continues with a specification of the procedures for creation of new class instances ([§12.5](#)); finalization of class instances ([§12.6](#)); and finalization of classes ([§12.7](#)). It concludes by describing the unloading of classes ([§12.8](#)) and the procedure followed when a virtual machine exits ([§12.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.1 Virtual Machine Start-Up

A Java Virtual Machine starts execution by invoking the method `main` of some specified class, passing it a single argument, which is an array of strings. In the examples in this specification, this first class is typically called `Test`.

The manner in which the initial class is specified to the Java Virtual Machine is beyond the scope of this specification, but it is typical, in host environments that use command lines, for the fully-qualified name of the class to be specified as a command-line argument and for following command-line arguments to be used as strings to be provided as the argument to the method `main`. For example, in a UNIX implementation, the command line:

```
java Test reboot Bob Dot Enzo
```

will typically start a Java Virtual Machine by invoking method `main` of class `Test` (a class in an unnamed package), passing it an array containing the four strings `"reboot"`, `"Bob"`, `"Dot"`, and `"Enzo"`.

We now outline the steps the virtual machine may take to execute `Test`, as an example of the loading, linking, and initialization processes that are described further in later sections.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.1.1 Load the Class Test

The initial attempt to execute the method `main` of class `Test` discovers that the class `Test` is not loaded—that is, that the virtual machine does not currently contain a binary representation for this class. The virtual machine then uses a class loader (§20.14) to attempt to find such a binary representation. If this process fails, then an error is thrown. This loading process is described further in §12.2.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.1.2 Link Test: Verify, Prepare, (Optionally) Resolve

After `Test` is loaded, it must be initialized before `main` can be invoked. And `Test`, like all (class or interface) types, must be linked before it is initialized. Linking involves verification, preparation and (optionally) resolution. Linking is described further in [§12.3](#).

Verification checks that the loaded representation of `Test` is well-formed, with a proper symbol table. Verification also checks that the code that implements `Test` obeys the semantic requirements of Java and the Java Virtual Machine. If a problem is detected during verification, then an error is thrown. Verification is described further in [§12.3.1](#).

Preparation involves allocation of static storage and any data structures that are used internally by the virtual machine, such as method tables. If a problem is detected during preparation, then an error is thrown. Preparation is described further in [§12.3.2](#).

Resolution is the process of checking symbolic references from `Test` to other classes and interfaces, by loading the other classes and interfaces that are mentioned and checking that the references are correct.

The resolution step is optional at the time of initial linkage. An implementation may resolve symbolic references from a class or interface that is being linked very early, even to the point of resolving all symbolic references from the classes and interfaces that are further referenced, recursively. (This resolution may result in errors from these further loading and linking steps.) This implementation choice represents one extreme and is similar to the kind of "static" linkage that has been done for many years in simple implementations of the C language. (In these implementations, a compiled program is typically represented as an "a.out" file that contains a fully-linked version of the program, including completely resolved links to library routines used by the program. Copies of these library routines are included in the "a.out" file.)

An implementation may instead choose to resolve a symbolic reference only when it is actively used; consistent use of this strategy for all symbolic references would represent the "laziest" form of resolution. In this case, if `Test` had several symbolic references to another class, then the references might be resolved one at a time, as they are used, or perhaps not at all, if these references were never used during execution of the program.

The only requirement on when resolution is performed is that any errors detected during resolution must be thrown at a point in the program where some action is taken by the program that might, directly or indirectly, require linkage to the class or interface involved in the error. Using the "static" example implementation choice described above, loading and linkage errors could occur before the program is executed if they involved a class or interface mentioned in the class `Test` or any of the further, recursively referenced, classes and interfaces. In a system that implemented the "laziest" resolution, these errors would be thrown only when an incorrect symbolic reference is actively used.

The resolution process is described further in [§12.3.3](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.1.3 Initialize Test: Execute Initializers

In our continuing example, the virtual machine is still trying to execute the method `main` of class `Test`. This is an attempted active use ([§12.4.1](#)) of the class, which is permitted only if the class has been initialized.

Initialization consists of execution of any class variable initializers and static initializers of the class `Test`, in textual order. But before `Test` can be initialized, its direct superclass must be initialized, as well as the direct superclass of its direct superclass, and so on, recursively. In the simplest case, `Test` has `Object` as its implicit direct superclass; if class `Object` has not yet been initialized, then it must be initialized before `Test` is initialized. Class `Object` has no superclass, so the recursion terminates here.

If class `Test` has another class `Super` as its superclass, then `Super` must be initialized before `Test`. This requires loading, verifying, and preparing `Super` if this has not already been done and, depending on the implementation, may also involve resolving the symbolic references from `Super` and so on, recursively.

Initialization may thus cause loading, linking, and initialization errors, including such errors involving other types.

The initialization process is described further in [§12.4](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.1.4 Invoke Test.main

Finally, after completion of the initialization for class `Test` (during which other consequential loading, linking, and initializing may have occurred), the method `main` of `Test` is invoked.

The method `main` must be declared `public`, `static`, and `void`. It must accept a single argument that is an array of strings.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.2 Loading of Classes and Interfaces

Loading refers to the process of finding the binary form of a class or interface type with a particular name, perhaps by computing it on the fly, but more typically by retrieving a binary representation previously computed from source code by a compiler, and constructing, from that binary form, a `Class` object to represent the class or interface.

The binary format of a class or interface is normally the `class` file format described in *The Java Virtual Machine*, but other formats are possible, provided they meet the requirements specified in [§13.1](#). The method `defineClass` ([§20.14.3](#)) of class `ClassLoader` may be used to construct `Class` objects from binary representations in the `class` file format.

A Java Virtual Machine system should maintain an internal table of classes and interfaces that have been loaded for the sake of resolving symbolic references. Each entry in the table should consist of a fully qualified class name (as a string), a class loader, and a `Class` object. Whenever a symbolic reference to a class or interface is to be resolved, a class loader is identified that is responsible for loading the class or interface, if necessary. The table should be consulted first, however; if it already contains an entry for that class name and class loader, then the class object in that entry should be used and no method of the class loader should be invoked. If the table contains no such entry, then the method `loadClass` ([§20.14.2](#)) of the class loader should be invoked, giving it the name of the class or interface. If and when it returns, the class object that it returns should be used to make a new entry in the table for that class name and class loader.

The purpose of this internal table is to allow the verification process ([§12.3.1](#)) to assume, for its purposes, that two classes or interfaces are the same if they have the same name and the same class loader. This property allows a class to be verified without loading all the classes and interfaces that it uses, whether actively or passively. Well-behaved class loaders do maintain this property: given the same name twice, a good class loader should return the same class object each time. But without the internal table, a malicious class loader could violate this property and undermine the security of the Java type system. A basic principle of the design of the Java language is that the type system cannot be subverted by code written in Java, not even by implementations of such otherwise sensitive system classes as `ClassLoader` ([§20.14](#)) and `SecurityManager` ([§20.17](#)).

An entry may be deleted from the internal table only after unloading ([§12.8](#)) the class or interface represented by the class object in the entry.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.2.1 The Loading Process

The loading process is implemented by the class `ClassLoader` (§20.14) and its subclasses. Different subclasses of `ClassLoader` may implement different loading policies. In particular, a class loader may cache binary representations of classes and interfaces, prefetch them based on expected usage, or load a group of related classes together. These activities may not be completely transparent to a running Java application if, for example, a newly compiled version of a class is not found because an older version is cached by a class loader. It is the responsibility of a class loader, however, to reflect loading errors only at points in the program they could have arisen without prefetching or group loading.

If an error occurs during class loading, then an instance of one of the following subclasses of class `LinkageError` will be thrown at any point in the Java program that (directly or indirectly) uses the type:

- `ClassCircularityError`: A class or interface could not be loaded because it would be its own superclass or superinterface (§13.4.4).
- `ClassFormatError`: The binary data that purports to specify a requested compiled class or interface is malformed.
- `NoClassDefFoundError`: No definition for a requested class or interface could be found by the relevant class loader.

Because loading involves the allocation of new data structures, it may fail with an `OutOfMemoryError`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.2.2 Implications for Code Generation

A cooperating class loader can enable a code generator to generate code for a group of class and interface types-perhaps an entire package-by loading the binary code for these types as a group. A format can be designed that allows all the internal symbolic references in such a group to be resolved, before the group is loaded. Such a strategy may also allow the generated code to be optimized before loading based on the known concrete types in the group. This approach may be useful in specific cases, but is discouraged as a general technique, since such a class file format is unlikely to be widely understood.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.3 Linking of Classes and Interfaces

Linking is the process of taking a binary form of a class or interface type and combining it into the runtime state of the Java Virtual Machine, so that it can be executed. A class or interface type is always loaded before it is linked. Three different activities are involved in linking: verification, preparation, and resolution of symbolic references.

Java allows an implementation flexibility as to when linking activities (and, because of recursion, loading) take place, provided that the semantics of the language are respected, that a class or interface is completely verified and prepared before it is initialized, and that errors detected during linkage are thrown at a point in the program where some action is taken by the program that might require linkage to the class or interface involved in the error.

For example, an implementation may choose to resolve each symbolic reference in a class or interface individually, only when it is used (lazy or late resolution), or to resolve them all at once while the class is being verified (static resolution). This means that the resolution process may continue, in some implementations, after a class or interface has been initialized.

Because linking involves the allocation of new data structures, it may fail with an `OutOfMemoryError`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.3.1 Verification of the Binary Representation

Verification ensures that the binary representation of a class or interface is structurally correct. For example, it checks that every instruction has a valid operation code; that every branch instruction branches to the start of some other instruction, rather than into the middle of an instruction; that every method is provided with a structurally correct signature; and that every instruction obeys the type discipline of the Java language.

For a more detailed description of the verification process, see the separate volume of this series, *The Java Virtual Machine Specification*.

If an error occurs during verification, then an instance of the following subclass of class `LinkageError` will be thrown at the point in the Java program that caused the class to be verified:

- `VerifyError`: The binary definition for a class or interface failed to pass a set of required checks to verify that it obeys the semantics of the Java language and that it cannot violate the integrity of the Java Virtual Machine. (See [§13.4.2](#), [§13.4.4](#), [§13.4.8](#), and [§13.4.16](#) for some examples.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.3.2 Preparation of a Class or Interface Type

Preparation involves creating the `static` fields (class variables and constants) for a class or interface and initializing such fields to the standard default values ([§4.5.4](#)). This does not require the execution of any Java code; explicit initializers for `static` fields are executed as part of initialization ([§12.4](#)), not preparation.

Java implementations must detect the following error during preparation:

- `AbstractMethodError`: A class that is not declared to be `abstract` has an `abstract` method. This can occur, for example, if a method that is originally not `abstract` is changed to be `abstract` after another class that inherits the now `abstract` method declaration has been compiled ([§13.4.15](#)).

If such an error is detected, then an instance of `AbstractMethodError` should be thrown at the point in the Java program that caused the class to be prepared.

Implementations of the Java Virtual Machine may precompute additional data structures at preparation time in order to make later operations on a class or interface more efficient. One particularly useful data structure is a "method table" or other data structure that allows any method to be invoked on instances of a class without requiring a search of superclasses at invocation time.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.3.3 Resolution of Symbolic References

A Java binary file references other classes and interfaces and their fields, methods, and constructors symbolically, using the fully-qualified names of the other classes and interfaces (§13.1). For fields and methods, these symbolic references include the name of the class or interface type that declares the field or method as well as the name of the field or method itself, together with appropriate type information.

Before a symbolic reference can be used it must be undergo *resolution*, wherein a symbolic reference is checked to be correct and, typically, replaced with a direct reference that can be more efficiently processed if the reference is used repeatedly.

If an error occurs during resolution, then an error will be thrown. Most typically, this will be an instance of one of the following subclasses of the class `IncompatibleClassChangeError`, but it may also be an instance of some other subclass of `IncompatibleClassChangeError` or even an instance of the class `IncompatibleClassChangeError` itself. This error may be thrown at any point in the program that uses a symbolic reference to the type, directly or indirectly:

- `IllegalAccessError`: A symbolic reference has been encountered that specifies a use or assignment of a field, or invocation of a method, or creation of an instance of a class, to which the code containing the reference does not have access because the field or method was declared `private`, `protected`, or default access (not `public`), or because the class was not declared `public`. This can occur, for example, if a field that is originally declared `public` is changed to be `private` after another class that refers to the field has been compiled (§13.4.6).
- `InstantiationError`: A symbolic reference has been encountered that is used in a class instance creation expression, but an instance cannot be created because the reference turns out to refer to an interface or to an `abstract` class. This can occur, for example, if a class that is originally not `abstract` is changed to be `abstract` after another class that refers to the class in question has been compiled (§13.4.1).
- `NoSuchFieldError`: A symbolic reference has been encountered that refers to a specific field of a specific class or interface, but the class or interface does not declare a field of that name (it is specifically not sufficient for it simply to be an inherited field of that class or interface). This can occur, for example, if a field declaration was deleted from a class after another class that refers to the field was compiled (§13.4.7).
- `NoSuchMethodError`: A symbolic reference has been encountered that refers to a specific method of a specific class or interface, but the class or interface does not declare a method of that signature (it is specifically not sufficient for it simply to be an inherited method of that class or interface). This can occur, for example, if a method declaration was deleted from a class after another class that refers to the method was compiled (§13.4.12).

Additionally, an `UnsatisfiedLinkError` (a subclass of `LinkageError`) may be thrown if a class declares a `native` method for which no implementation can be found. The error will occur if the method is used, or earlier depending on what kind of resolution strategy is being used by the virtual machine (§12.3).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.3.4 Implications for Code Generation

The symbolic references within a group of types may be resolved even before the group is loaded (§12.2.2), in an implementation that uses a special (non-standard) binary format (§13.1). This corresponds to the traditional practice of "linkage editing." Even if this is not done, a Java implementation has a lot of flexibility. It may resolve all symbolic references from a type at the point of the first linkage activity on the type, or defer the resolution of each symbolic reference to the first use of that reference.

We note that the flexibility accorded the Java implementation in the linkage process does not affect correctly formed Java programs, which should never encounter linkage errors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.4 Initialization of Classes and Interfaces

Initialization of a class consists of executing its static initializers and the initializers for `static` fields (class variables) declared in the class. Initialization of an interface consists of executing the initializers for fields (constants) declared there.

Before a class is initialized, its superclass must be initialized, but interfaces implemented by the class need not be initialized. Similarly, the superinterfaces of an interface need not be initialized before the interface is initialized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.4.1 When Initialization Occurs

A class or interface type *T* will be *initialized* at its first *active use*, which occurs if:

- *T* is a class and a method actually declared in *T* (rather than inherited from a superclass) is invoked.
- *T* is a class and a constructor for class *T* is invoked, or *U* is an array with element type *T*, and an array of type *U* is created.
- A non-constant field declared in *T* (rather than inherited from a superclass or superinterface) is used or assigned. A *constant field* is one that is (explicitly or implicitly) both `final` and `static`, and that is initialized with the value of a compile-time constant expression (§15.27). Java specifies that a reference to a constant field must be resolved at compile time to a copy of the compile-time constant value, so uses of such a field are never active uses. See §13.4.8 for a further discussion.

All other uses of a type are *passive uses*.

The intent here is that a class or interface type has a set of initializers that put it in a consistent state, and that this state is the first state that is observed by other classes. The static initializers and class variable initializers are executed in textual order, and may not refer to class variables declared in the class whose declarations appear textually after the use, even though these class variables are in scope (§8.5). This restriction is designed to detect, at compile time, most circular or otherwise malformed initializations.

As shown in an example in §8.5, the fact that initialization code is unrestricted allows examples to be constructed where the value of a class variable can be observed when it still has its initial default value, before its initializing expression is evaluated, but such examples are rare in practice. (Such examples can be also constructed for instance variable initialization; see the example at the end of §12.5). Java provides the full power of the language in these initializers; programmers must exercise some care. This power places an extra burden on code generators, but this burden would arise in any case because Java is concurrent (§12.4.3).

Before a class is initialized, its superclasses are initialized, if they have not previously been initialized.

Thus, the test program:

```
class Super {
    static { System.out.print("Super "); }
}

class One {
    static { System.out.print("One "); }
}

class Two extends Super {
    static { System.out.print("Two "); }
}

class Test {
    public static void main(String[] args) {
        One o = null;
        Two t = new Two();
        System.out.println((Object)o == (Object)t);
    }
}
```



```
    }  
}
```

prints:

```
Super Two false
```

The class `One` is never initialized, because it not used actively and therefore is never linked to. The class `Two` is initialized only after its superclass `Super` has been initialized.

A reference to a field is an active use of only the class or interface that actually declares it, even though it might be referred to through the name of a subclass, a subinterface, or a class that implements an interface. The test program:

```
class Super { static int taxi = 1729; }  
  
class Sub extends Super {  
    static { System.out.print("Sub "); }  
}  
  
class Test {  
    public static void main(String[] args) {  
        System.out.println(Sub.taxi);  
    }  
}
```

prints only:

```
1729
```

because the class `Sub` is never initialized; the reference to `Sub.taxi` is a reference to a field actually declared in class `Super` and is not an active use of the class `Sub`.

Initialization of an interface does not, of itself, require initialization of any of its superinterfaces. Thus, the test program:

```
interface I {  
    int i = 1, ii = Test.out("ii", 2);  
}  
  
interface J extends I {  
    int j = Test.out("j", 3), jj = Test.out("jj", 4);  
}  
  
interface K extends J {  
    int k = Test.out("k", 5);  
}  
  
class Test {
```



```

    public static void main(String[] args) {
        System.out.println(J.i);
        System.out.println(K.j);
    }

    static int out(String s, int i) {
        System.out.println(s + "=" + i);
        return i;
    }
}

```

produces the output:

```

1
j=3
jj=4
3

```

The reference to `J.i` is to a field that is a compile-time constant; therefore, it does not cause `I` to be initialized. The reference to `K.j` is a reference to a field actually declared in interface `J` that is not a compile-time constant; this causes initialization of the fields of interface `J`, but not those of its superinterface `I`, nor those of interface `K`. Despite the fact that the name `K` is used to refer to field `j` of interface `J`, interface `K` is not actively used.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.4.2 Detailed Initialization Procedure

Because Java is multithreaded, initialization of a class or interface requires careful synchronization, since some other thread may be trying to initialize the same class or interface at the same time. There is also the possibility that initialization of a class or interface may be requested recursively as part of the initialization of that class or interface; for example, a variable initializer in class *A* might invoke a method of an unrelated class *B*, which might in turn invoke a method of class *A*. The implementation of the Java Virtual Machine is responsible for taking care of synchronization and recursive initialization by using the following procedure. It assumes that the `Class` object has already been verified and prepared, and that the `Class` object contains state that indicates one of four situations:

- This `Class` object is verified and prepared but not initialized.
- This `Class` object is being initialized by some particular thread *T*.
- This `Class` object is fully initialized and ready for use.
- This `Class` object is in an erroneous state, perhaps because the verification or preparation step failed, or because initialization was attempted and failed.

The procedure for initializing a class or interface is then as follows:

Synchronize (§14.17) on the `Class` object that represents the class or interface to be initialized.

This involves waiting until the current thread can obtain the lock for that object (§17.13).

If initialization is in progress for the class or interface by some other thread, then wait (§20.1.6) on this `Class` object (which temporarily releases the lock). When the current thread awakens from the wait, repeat this step.

If initialization is in progress for the class or interface by the current thread, then this must be a recursive request for initialization. Release the lock on the `Class` object and complete normally.

If the class or interface has already been initialized, then no further action is required. Release the lock on the `Class` object and complete normally.

If the `Class` object is in an erroneous state, then initialization is not possible. Release the lock on the `Class` object and throw a `NoClassDefFoundError`.

Otherwise, record the fact that initialization of the `Class` object is now in progress by the current thread and release the lock on the `Class` object.

Next, if the `Class` object represents a class rather than an interface, and the superclass of this class has not yet been initialized, then recursively perform this entire procedure for the superclass. If necessary, verify and prepare the superclass first. If the initialization of the superclass completes abruptly because of a thrown exception, then lock this `Class` object, label it erroneous, notify all waiting threads (§20.1.10), release the lock, and complete abruptly, throwing the same exception that resulted from initializing the superclass.

Next, execute either the class variable initializers and static initializers of the class, or the field initializers of the interface, in textual order, as though they were a single block, except that `final` class variables and fields of interfaces whose values are compile-time constants are initialized first (§8.3.2.1, §9.3.1, §13.4.8).

If the execution of the initializers completes normally, then lock this `Class` object, label it fully initialized, notify all waiting threads (§20.1.10), release the lock, and complete this procedure normally.

Otherwise, the initializers must have completed abruptly by throwing some exception *E*. If the class of *E* is not `Error` or one of its subclasses, then create a new instance of the class `ExceptionInInitializerError`, with *E* as the argument, and use this object in place of *E* in the following step. But if a new instance of `ExceptionInInitializerError` cannot be

created because an `OutOfMemoryError` occurs, then instead use an `OutOfMemoryError` object in place of *E* in the following step.

Lock the `Class` object, label it erroneous, notify all waiting threads ([§20.1.10](#)), release the lock, and complete this procedure abruptly with reason *E* or its replacement as determined in the previous step.

(Due to a flaw in some early implementations of Java, a exception during class initialization was ignored, rather than causing an `ExceptionInInitializerError` as described here.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.4.3 Implications for Code Generation

Code generators need to preserve the points of possible initialization of a class or interface, inserting an invocation of the initialization procedure just described. If this initialization procedure completes normally and the `Class` object is fully initialized and ready for use, then the invocation of the initialization procedure is no longer necessary and it may be eliminated from the code—for example, by patching it out or otherwise regenerating the code.

Compile-time analysis may, in some cases, be able to eliminate many of the checks that a type has been initialized from the generated code, if an initialization order for a group of related types can be determined. Such analysis must, however, fully account for the fact that Java is concurrent and that initialization code is unrestricted.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.5 Creation of New Class Instances

A new class instance is explicitly created when one of the following situations occurs:

- Evaluation of a class instance creation expression (§15.8) creates a new instance of the class whose name appears in the expression.
- Invocation of the `newInstance` method (§20.3.6) of class `Class` creates a new instance of the class represented by the `Class` object for which the method was invoked.

A new class instance may be implicitly created in the following situations:

- Loading of a class or interface that contains a `String` literal (§3.10.5) may create a new `String` object (§20.12) to represent that literal. (This might not occur if the same `String` has previously been interned (§3.10.5).)
- Execution of a string concatenation operator (§15.17.1) that is not part of a constant expression sometimes creates a new `String` object to represent the result. String concatenation operators may also create temporary wrapper objects for a value of a primitive type.

Each of these situations identifies a particular constructor to be called with specified arguments (possibly none) as part of the class instance creation process.

Whenever a new class instance is created, memory space is allocated for it with room for all the instance variables declared in the class type and all the instance variables declared in each superclass of the class type, including all the instance variables that may be hidden. If there is not sufficient space available to allocate memory for the object, then creation of the class instance completes abruptly with an `OutOfMemoryError`. Otherwise, all the instance variables in the new object, including those declared in superclasses, are initialized to their default values (§4.5.4). Just before a reference to the newly created object is returned as the result, the indicated constructor is processed to initialize the new object using the following procedure:

Assign the arguments for the constructor to newly created parameter variables for this constructor invocation.

If this constructor begins with an explicit constructor invocation of another constructor in the same class (using `this`), then evaluate the arguments and process that constructor invocation recursively using these same five steps. If that constructor invocation completes abruptly, then this procedure completes abruptly for the same reason; otherwise, continue with step 5.

This constructor does not begin with an explicit constructor invocation of another constructor in the same class (using `this`). If this constructor is for a class other than `Object`, then this constructor will begin with a explicit or implicit invocation of a superclass constructor (using `super`). Evaluate the arguments and process that superclass constructor invocation recursively using these same five steps. If that constructor invocation completes abruptly, then this procedure completes abruptly for the same reason. Otherwise, continue with step 4.

Execute the instance variable initializers for this class, assigning their values to the corresponding instance variables, in the left-to-right order in which they appear textually in the source code for the class. If execution of any of these initializers results in an exception, then no further initializers are processed and this procedure completes abruptly with that same exception. Otherwise, continue with step 5. (In some early Java implementations, the compiler incorrectly omitted the code to initialize a field if the field initializer expression was a constant expression whose value was equal to the default initialization value for its type.)

Execute the rest of the body of this constructor. If that execution completes abruptly, then this procedure completes abruptly for the same reason. Otherwise, this procedure completes normally.

In the example:

```
class Point {
    int x, y;
    Point() { x = 1; y = 1; }
}

class ColoredPoint extends Point {
    int color = 0xFF00FF;
}

class Test {
    public static void main(String[] args) {
        ColoredPoint cp = new ColoredPoint();
        System.out.println(cp.color);
    }
}
```

a new instance of `ColoredPoint` is created. First, space is allocated for the new `ColoredPoint`, to hold the fields `x`, `y`, and `color`. All these fields are then initialized to their default values (in this case, 0 for each field). Next, the `ColoredPoint` constructor with no arguments is first invoked. Since `ColoredPoint` declares no constructors, a default constructor of the form:

```
ColoredPoint() { super(); }
```

is provided for it automatically by the Java compiler.

This constructor then invokes the `Point` constructor with no arguments. The `Point` constructor does not begin with an invocation of a constructor, so the compiler provides an implicit invocation of its superclass constructor of no arguments, as though it had been written:

```
Point() { super(); x = 1; y = 1; }
```

Therefore, the constructor for `Object` which takes no arguments is invoked.

The class `Object` has no superclass, so the recursion terminates here. Next, any instance variable initializers and static initializers of `Object` are invoked. Next, the body of the constructor of `Object` that takes no arguments is executed. No such constructor is declared in `Object`, so the compiler supplies a default one, which in this special case is:

```
Object() { }
```

This constructor executes without effect and returns.

Next, all initializers for the instance variables of class `Point` are executed. As it happens, the declarations of `x` and `y` do not provide any initialization expressions, so no action is required for this step of the example. Then the body of the `Point` constructor is executed, setting `x` to 1 and `y` to 1.

Next, the initializers for the instance variables of class `ColoredPoint` are executed. This step assigns the value `0xFF00FF` to `color`. Finally, the rest of the body of the `ColoredPoint`

constructor is executed (the part after the invocation of `super`); there happen to be no statements in the rest of the body, so no further action is required and initialization is complete.

Unlike C++, the Java language does not specify altered rules for method dispatch during the creation of a new class instance. If methods are invoked that are overridden in subclasses in the object being initialized, then these overriding methods are used, even before the new object is completely initialized. Thus, compiling and running the example:

```
class Super {
    Super() { printThree(); }
    void printThree() { System.out.println("three"); }
}
```

```
class Test extends Super {
    int indiana = (int)Math.PI;           // That is, 3

    public static void main(String[] args) {
        Test t = new Test();
        t.printThree();
    }
    void printThree() { System.out.println(indiana); }
}
```

produces the output:

```
0
3
```

This shows that the invocation of `printThree` in the constructor for class `Super` does not invoke the definition of `printThree` in class `Super`, but rather invokes the overriding definition of `printThree` in class `Test`. This method therefore runs before the field initializers of `Test` have been executed, which is why the first value output is 0, the default value to which the field `three` of `Test` is initialized. The later invocation of `printThree` in method `main` invokes the same definition of `printThree`, but by that point the initializer for instance variable `three` has been executed, and so the value 3 is printed.

See [§8.6](#) for more details on constructor declarations.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

12.6 Finalization of Class Instances

The class `Object` has a protected method called `finalize` (§20.1.11); this method can be overridden by other classes. The particular definition of `finalize` that can be invoked for an object is called the *finalizer* of that object. Before the storage for an object is reclaimed by the garbage collector, the Java Virtual Machine will invoke the finalizer of that object.

Finalizers provide a chance to free up resources (such as file descriptors or operating system graphics contexts) that cannot be freed automatically by an automatic storage manager. In such situations, simply reclaiming the memory used by an object would not guarantee that the resources it held would be reclaimed.

The Java language does not specify how soon a finalizer will be invoked, except to say that it will happen before the storage for the object is reused. Also, the Java language does not specify which thread will invoke the finalizer for any given object. If an uncaught exception is thrown during the finalization, the exception is ignored and finalization of that object terminates.

The `finalize` method declared in class `Object` takes no action. However, the fact that class `Object` declares a `finalize` method means that the `finalize` method for any class can always invoke the `finalize` method for its superclass, which is usually good practice. (Unlike constructors, finalizers do not automatically invoke the finalizer for the superclass; such an invocation must be coded explicitly.)

For efficiency, an implementation may keep track of classes that do not override the `finalize` method of class `Object`, or override it in a trivial way, such as:

```
protected void finalize() throws Throwable {  
    super.finalize();  
}
```

We encourage implementations to treat such objects as having a finalizer that is not overridden, and to finalize them more efficiently, as described in §12.6.1.

A finalizer may be invoked explicitly, just like any other method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.6.1 Implementing Finalization

Every object can be characterized by two attributes: it may be *reachable*, *finalizer-reachable*, or *unreachable*, and it may also be *unfinalized*, *finalizable*, or *finalized*.

A *reachable* object is any object that can be accessed in any potential continuing computation from any live thread. Optimizing transformations of a program can be designed that reduce the number of objects that are reachable to be less than those which would naively be considered reachable. For example, a compiler or code generator may choose, explicitly or implicitly, to set a variable or parameter that will no longer be used to `null` to cause the storage for such an object to be potentially reclaimable sooner. A *finalizer-reachable* object can be reached from some finalizable object through some chain of references, but not from any live thread. An *unreachable* object cannot be reached by either means.

An *unfinalized* object has never had its finalizer automatically invoked; a *finalized* object has had its finalizer automatically invoked. A *finalizable* object has never had its finalizer automatically invoked, but the Java Virtual Machine may eventually automatically invoke its finalizer.

The life cycle of an object obeys the following transition diagram, where we abbreviate "finalizer-reachable" as "f-reachable":

```
{ewc msdncd, EWGraphic, LNG20p 0 /a "langref2ANC.BMP"}
```

When an object is first created (A), it is reachable and unfinalized.

As references to an object are discarded during program execution, an object that was reachable may become finalizer-reachable (B, C, D) or unreachable (E, F). (Note that a finalizer-reachable object never becomes unreachable directly; it becomes reachable when the finalizer from which it can be reached is invoked, as explained below.)

If the Java Virtual Machine detects that an unfinalized object has become finalizer-reachable or unreachable, it may label the object finalizable (G, H); moreover, if the object was unreachable, it becomes finalizer-reachable (H).

If the Java Virtual Machine detects that a finalized object has become unreachable, it may reclaim the storage occupied by the object because the object will never again become reachable (I).

At any time, a Java Virtual Machine may take any finalizable object, label it finalized, and then invoke its `finalize` method in some thread. This causes the object to become finalized and reachable (J, K), and it also may cause other objects that were finalizer-reachable to become reachable again (L, M, N).

A finalizable object cannot also be unreachable; it can be reached because its finalizer may eventually be invoked, whereupon the thread running the finalizer will have access to the object, as [this \(§15.7.2\)](#). Thus, there are actually only eight possible states for an object.

After an object has been finalized, no further action is taken until the automatic storage management determines that it is unreachable. Because of the way that an object progresses from the *unfinalized* state through the *finalizable* state to the *finalized* state, the `finalize` method is never automatically invoked more than once by a Java Virtual Machine for each object, even if the object is again made reachable after it has been finalized.

Explicit invocation of a finalizer ignores the current state of the object and does not change the state of the object from unfinalized or finalizable to finalized.

If a class does not override method `finalize` of class `Object` (or overrides it in only a trivial way, as described above), then if instances of such as class become unreachable, they may be discarded immediately rather than made to await a second determination that they have become unreachable. This strategy is indicated by the dashed arrow (O) in the transition diagram.

Java programmers should also be aware that a finalizer can be automatically invoked, even though it

is reachable, during finalization-on-exit ([§12.9](#)); moreover, a finalizer can also be invoked explicitly as an ordinary method. Therefore, we recommend that the design of `finalize` methods be kept simple and that they be programmed defensively, so that they will work in all cases.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.6.2 Finalizer Invocations are Not Ordered

Java imposes no ordering on finalize method calls. Finalizers may be called in any order, or even concurrently.

As an example, if a circularly linked group of unfinalized objects becomes unreachable (or finalizer-reachable), then all the objects may become finalizable together. Eventually, the finalizers for these objects may be invoked, in any order, or even concurrently using multiple threads. If the automatic storage manager later finds that the objects are unreachable, then their storage can be reclaimed.

It is straightforward to implement a Java class that will cause a set of finalizer-like methods to be invoked in a specified order for a set of objects when all the objects become unreachable. Defining such a class is left as an exercise for the reader.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.7 Finalization of Classes

If a class declares a class method `classFinalize` that takes no arguments and returns no result:

```
static void classFinalize() throws Throwable { . . . }
```

then this method will be invoked before the class is unloaded ([§12.8](#)). Like the `finalize` method for objects, this method will be automatically invoked only once. This method may optionally be declared `private`, `protected`, or `public`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.8 Unloading of Classes and Interfaces

A Java Virtual Machine may provide mechanisms whereby classes are *unloaded*. The details of such mechanisms are not specified in this version of the Java Language Specification. In general, groups of related class and interface types will be unloaded together. This can be used, for example, to unload a group of related types that have been loaded using a particular class loader. Such a group might consist of all the classes implementing a single applet in a Java-based browser such as HotJava, for example.

A class may not be unloaded while any instance of it is still reachable ([§12.6](#)). A class or interface may not be unloaded while the `Class` object that represents it is still reachable.

Classes that declare class finalizers ([§12.7](#)) will have these finalizers run before they are unloaded.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


12.9 Virtual Machine Exit

A Java Virtual Machine terminates all its activity and *exits* when one of two things happens:

- All the threads that are not daemon threads ([§20.20.24](#)) terminate.
- Some thread invokes the `exit` method ([§20.16.2](#)) of class `Runtime` or class `System` and the exit operation is not forbidden by the security manager ([§20.17.13](#)).

A Java program can specify that the finalizers of all objects that have finalizers, and all classes that have class finalizers, that have not yet been automatically invoked are to be run before the virtual machine exits. This is done by invoking the method `runFinalizersOnExit` of class `System` with the argument `true`. The default is to not run finalizers on exit, and this behavior may be restored by invoking `runFinalizersOnExit` with the argument `false`. An invocation of the `runFinalizersOnExit` method is permitted only if the caller is allowed to `exit`, and is otherwise rejected by the `SecurityManager` ([§20.17](#)).

... Farewell!

The day frowns more and more. Thou'rt like to have

A lullaby too rough: I never saw

The heavens so dim by day: A savage clamour!

Well may I get aboard! This is the chase.

I am gone for ever! [Exit, pursued by a bear]

--William Shakespeare, *The Winter's Tale*, Act III, scene iii

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Binary Compatibility

Java development tools should support automatic recompilation as necessary whenever source code is available. Particular implementations of Java may also store the source and binary of types in a versioning database and implement a `ClassLoader` (§20.14) that uses integrity mechanisms of the database to prevent linkage errors by providing binary-compatible versions of types to clients.

Developers of packages and classes that are to be widely distributed face a different set of problems. In the Internet, which is our favorite example of a widely distributed system, it is often impractical or impossible to automatically recompile the pre-existing binaries that directly or indirectly depend on a type that is to be changed. Instead, Java defines a set of changes that developers are permitted to make to a package or to a class or interface type while preserving (not breaking) compatibility with existing binaries.

The paper quoted above appears in *Proceedings of OOPSLA '95*, published as *ACM SIGPLAN Notices*, Volume 30, Number 10, October 1995, pages 426-438. Within the framework of that paper, Java binaries are binary (release-to-release) compatible under all relevant transformations that the authors identify. Using their scheme, here is a list of some important binary compatible changes that Java supports:

- Reimplementing existing methods, constructors, and initializers to improve performance.
- Changing methods or constructors to return values on inputs for which they previously either threw exceptions that normally should not occur or failed by going into an infinite loop or causing a deadlock.
- Adding new fields, methods, or constructors to an existing class or interface.
- Deleting `private` fields, methods, or constructors of a class or interface.
- When an entire package is updated, deleting default (package-only) access fields, methods, or constructors of classes and interfaces in the package.
- Reordering the fields, methods, or constructors in an existing type declaration.
- Moving a method upward in the class hierarchy, provided a forwarding method is left in its place.
- Reordering the list of direct superinterfaces of a class or interface.
- Inserting new class or interface types in the type hierarchy.

This chapter specifies minimum standards for binary compatibility guaranteed by all Java implementations. Java guarantees compatibility when binaries of classes and interfaces are mixed that are not known to be from compatible sources, but whose sources have been modified in the compatible ways described here.

We encourage Java development systems to provide facilities that alert developers to the impact of changes on pre-existing binaries that cannot be recompiled.

This chapter first specifies some properties that any Java binary format must have (§13.1). It next defines binary compatibility, explaining what it is and what it is not (§13.2). It finally enumerates a large set of possible changes to packages (§13.3), classes (§13.4) and interfaces (§13.5), specifying which changes are guaranteed to preserve binary compatibility and which are not.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.1 The Form of a Java Binary

While many Java binary files are likely to be in exactly the `class` file format specified by the *The Java Virtual Machine Specification*, this specification does not mandate the use of any specific binary file format. Rather, it specifies properties that any binary format for compiled types must obey. A number of these properties are specifically chosen to support source code transformations that preserve binary compatibility.

The requirements are:

- Binary formats for Java programs must be defined and processed to respect the specifications of loading (§12.2), linking (§12.3) and initialization (§12.4) of class and interface types.
- A reference to another class or interface type must be symbolic, using the fully qualified name of the type as determined at compile time.
- A reference to a field of another class or interface must be resolved at compile time to a symbolic reference to the class or interface in which the field is declared, plus the simple name of the field. (Including the exact class or interface in which the field is declared makes the binaries more robust, since adding another field with the same name, even in a subclass, cannot cause confusion at link time. This rule does mean, however, that moving a field to a superclass is not a binary compatible change; see §13.4.5 for a discussion.) The reference must also include a symbolic reference to the declared type of the field so that the verifier can check that the type is as expected. References to fields that are `static`, `final`, and initialized with compile-time constant expressions are resolved at compile time to the constant value that is denoted. No reference to such a constant field should be present in the code in a binary file (except in the class or interface containing the constant field, which will have code to initialize it), and such constant fields must always appear to have been initialized; the default initial value for the type of such a field must never be observed. See §13.4.8 for a discussion.
- A reference to a method or constructor must be resolved at compile time to a symbolic reference to the class or interface in which the denoted method or constructor is declared, plus the signature of the method or constructor. (As for fields, this makes the binaries more robust, with the caveat that such a method cannot be moved to a superclass without leaving a forwarding method behind; see §13.4.5 for a discussion.) A reference to a method must also include either a symbolic reference to the return type of the denoted method or an indication that the denoted method is declared `void` and does not return a value. The signature of a method must include all of the following:
 - The simple name of the method
 - The number of parameters to the method
 - A symbolic reference to the type of each parameter

The signature of a constructor must include both:

- The number of parameters to the constructor
- A symbolic reference to the type of each parameter

A Java binary representation for a class or interface must also contain all of the following:

- If it is a class and is not class `java.lang.Object`, then a symbolic reference to the direct superclass of this class
- A symbolic reference to each direct superinterface, if any
- A specification of each field that is not `private` declared in the class or interface, given as the simple name of the field and a symbolic reference to the type of the field

- If it is a class, then the signature of each constructor, as described above
- For each method that is not `private` declared in the class or interface, its signature and return type, as described above
- The code needed to implement the class or interface:
 - For an interface, code for the field initializers
 - For a class, code for the field initializers, the static initializers, and the implementation of each method or constructor that is not declared `private`

If a Java system defines a binary format that represents a group of classes and interfaces comprised by an entire package, then this binary format need not expose information about fields, methods, or constructors that are declared with default (package) access.

The following sections specify the changes that may be made to class and interface type declarations without breaking compatibility with pre-existing binaries. The Java Virtual Machine and its standard `class` file format support these changes; other Java binary formats are required to support these changes as well.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.2 What Binary Compatibility Is and Is Not

A change to a type is *binary compatible with* (equivalently, does not *break binary compatibility* with) preexisting binaries if preexisting binaries that previously linked without error will continue to link without error.

As described in [§13.1](#), symbolic references to methods and fields name the exact class or interface in which the method or field is declared. This means that binaries are compiled to rely on the accessible members and constructors of other classes and interfaces. To preserve binary compatibility, a class or interface should treat these accessible members and constructors, their existence and behavior, as a *contract* with users of the class or interface.

Java is designed to prevent additions to contracts and accidental name collisions from breaking binary compatibility; specifically:

- Introducing a new field with the same name as an existing field, in a subclass of the class containing the existing field declaration, does not break compatibility with preexisting binaries. See the example at the beginning of [§13.4.5](#).
- Addition of more methods overloading a particular method name does not break compatibility with preexisting binaries. The method signature that the preexisting binary will use for method lookup is chosen by Java's method overload resolution algorithm at compile time ([§15.11.2](#)). (If Java had been designed so that the particular method to be executed was chosen at run time, then such an ambiguity might be detected at run time. Such a rule would imply that adding an additional overloaded method so as to make ambiguity possible at a call site became possible could break compatibility with an unknown number of preexisting binaries. See [§13.4.22](#) for more discussion.)

Binary compatibility is not the same as source compatibility. In particular, the example in [§13.4.5](#) shows that a set of compatible binaries can be produced from sources that will not compile all together. This example is typical: a new declaration is added, changing the meaning of a name in an unchanged part of the source code, while the preexisting binary for that unchanged part of the source code retains the fully-qualified, previous meaning of the name. Producing a consistent set of source code requires providing a qualified name or field access expression corresponding to the previous meaning.

We hope to make some improvements to future versions of Java to better support both source and binary compatible evolution of types. In particular, we are considering a mechanism to allow a class to implement two interfaces that have methods with the same signature but are to be considered different or have different return types. We welcome suggestions and proposals that would help us to make additional improvements, either in managing name and signature conflicts or other sources of incompatibility.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.3 Evolution of Packages

A new class or interface type may be added to a package without breaking compatibility with pre-existing binaries, provided the new type does not reuse a name previously given to an unrelated type. If a new type reuses a name previously given to an unrelated type, then a conflict may result, since binaries for both types could not be loaded by the same class loader.

Changes in class and interface types that are not `public` and that are not a superclass or superinterface, respectively, of a `public` type, affect only types within the package in which they are declared. Such types may be deleted or otherwise changed, even if incompatibilities are otherwise described here, provided that the affected binaries of that package are updated together.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4 Evolution of Classes

This section describes the effects of changes to the declaration of a class and its members and constructors on pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.1 abstract Classes

If a class that was not `abstract` is changed to be declared `abstract`, then pre-existing binaries that attempt to create new instances of that class will throw either an `InstantiationError` at link time, or an `InstantiationException` at run time (if the method `newInstance` ([§20.3.6](#)) of class `Class` is used); such a change is therefore not recommended for widely distributed classes.

Changing a class that was declared `abstract` to no longer be declared `abstract` does not break compatibility with pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.2 final Classes

If a class that was not declared `final` is changed to be declared `final`, then a `VerifyError` is thrown if a binary of a pre-existing subclass of this class is loaded, because `final` classes can have no subclasses; such a change is not recommended for widely distributed classes.

Changing a class that was declared `final` to no longer be declared `final` does not break compatibility with pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.3 public Classes

Changing a class that was not declared `public` to be declared `public` does not break compatibility with pre-existing binaries.

If a class that was declared `public` is changed to not be declared `public`, then an `IllegalAccessError` is thrown if a pre-existing binary is linked that needs but no longer has access to the class type; such a change is not recommended for widely distributed classes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.4 Superclasses and Superinterfaces

A `ClassCircularityError` is thrown at load time if a class would be a superclass of itself. Changes to the class hierarchy that could result in such a circularity when newly compiled binaries are loaded with pre-existing binaries are not recommended for widely distributed classes.

Changing the direct superclass or the set of direct superinterfaces of a class type will not break compatibility with pre-existing binaries, provided that the total set of superclasses or superinterfaces, respectively, of the class type loses no members.

Changes to the set of superclasses of a class will not break compatibility with pre-existing binaries simply because of uses of class variables and class methods. This is because uses of class variables and class methods are resolved at compile time to symbolic references to the name of the class that declares them. Such uses therefore depend only on the continuing existence of the class declaring the variable or method, not on the shape of the class hierarchy.

If a change to the direct superclass or the set of direct superinterfaces results in any class or interface no longer being a superclass or superinterface, respectively, then link-time errors may result if pre-existing binaries are loaded with the binary of the modified class. Such changes are not recommended for widely distributed classes. The resulting errors are detected by the verifier of the Java Virtual Machine when an operation that previously compiled would violate the type system. For example, suppose that the following test program:

```
class Hyper { char h = 'h'; }
class Super extends Hyper { char s = 's'; }
class Test extends Super {
    public static void main(String[] args) {
        Hyper h = new Super();
        System.out.println(h.h);
    }
}
```

is compiled and executed, producing the output:

h

Suppose that a new version of class `Super` is then compiled:

```
class Super { char s = 's'; }
```

This version of class `Super` is not a subclass of `Hyper`. If we then run the existing binaries of `Hyper` and `Test` with the new version of `Super`, then a `VerifyError` is thrown at link time. The verifier objects because the result of `new Super()` cannot be assigned to a variable of type `Hyper`, because `Super` is not a subclass of `Hyper`.

It is instructive to consider what might happen without the verification step: the program might run and print:

s

This demonstrates that without the verifier the type system could be defeated by linking inconsistent binary files, even though each was produced by a correct Java compiler.

As a further example, here is an implementation of a cast from a reference type to `int`, which could be made to run in certain implementations of Java if they failed to perform the verification process. Assume an implementation that uses method dispatch tables and whose linker assigns offsets into those tables in a sequential and straightforward manner. Then suppose that the following Java code is compiled:

```
class Hyper { int zero(Object o) { return 0; } }
class Super extends Hyper { int peek(int i) { return i; } }

class Test extends Super {
    public static void main(String[] args) throws Throwable {
        Super as = new Super();
        System.out.println(as);
        System.out.println(Integer.toHexString(as.zero(as)));
    }
}
```

The assumed implementation determines that the class `Super` has two methods: the first is method `zero` inherited from class `Hyper`, and the second is the method `peek`. Any subclass of `Super` would also have these same two methods in the first two entries of its method table. (Actually, all these methods would be preceded in the method tables by all the methods inherited from class `Object` but, to simplify the discussion, we ignore that here.) For the method invocation `as.zero(as)`, the compiler specifies that the first method of the method table should be invoked; this is always correct if type safety is preserved.

If the compiled code is then executed, it prints something like:

```
Super@ee300858
0
```

which is the correct output. But if a new version of `Super` is compiled, which is the same except for the `extends` clause:

```
class Super { int peek(int i) { return i; } }
```

then the first method in the method table for `Super` will now be `peek`, not `zero`. Using the new binary code for `Super` with the old binary code for `Hyper` and `Test` will cause the method invocation `as.zero(as)` to dispatch to the method `peek` in `Super`, rather than the method `zero` in `Hyper`. This is a type violation, of course; the argument is of type `Super` but the parameter is of type `int`. With a few plausible assumptions about internal data representations and the consequences of the type violation, execution of this incorrect program might produce the output:

Super@ee300848
ee300848

A `poke` method, capable of altering any location in memory, could be concocted in a similar manner. This is left as an exercise for the reader.

The lesson is that a implementation of Java that lacks a verifier or fails to use it will not maintain type safety and is, therefore, not a valid Java implementation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.5 Class Body and Member Declarations

No incompatibility with pre-existing binaries is caused by adding a class member that has the same name (for fields) or same name, signature, and return type (for methods) as a member of a superclass or subclass. References to the original field or method were resolved at compile time to a symbolic reference containing the name of the class in which they were declared. This makes compiled Java code more robust against changes than it might otherwise be. No error occurs even if the set of classes being linked would encounter a compile-time error. As an example, if the program:

```
class Hyper { String h = "Hyper"; }
class Super extends Hyper { }
class Test extends Super {
    public static void main(String[] args) {
        String s = new Test().h;
        System.out.println(s);
    }
}
```

is compiled and executed, it produces the output:

Hyper

Suppose that a new version of class `Super` is then compiled:

```
class Super extends Hyper { char h = 'h'; }
```

If the resulting binary is used with the existing binaries for `Hyper` and `Test`, then the output is still:

Hyper

even though compiling the source for these binaries:

```
class Hyper { String h = "Hyper"; }
class Super extends Hyper { char h = 'h'; }
class Test extends Super {
    public static void main(String[] args) {
        String s = new Test().h;
        System.out.println(s);
    }
}
```

would result in a compile-time error, because the `h` in the source code for `main` would now be construed as referring to the `char` field declared in `Super`, and a `char` value can't be assigned to a `String`.

Deleting a class member or constructor that is not declared `private` may cause a linkage error if the member or constructor is used by a pre-existing binary, even if the member was an instance

method that was overriding a superclass method. This is because, during resolution, the linker looks only in the class that was identified at compile time. Thus, if the program:

```
class Hyper {
    void hello() { System.out.println("hello from Hyper"); }
}

class Super extends Hyper {
    void hello() { System.out.println("hello from Super"); }
}

class Test {
    public static void main(String[] args) {
        new Super().hello();
    }
}
```

is compiled and executed, it produces the output:

```
hello from Super
```

Suppose that a new version of class `Super` is produced:

```
class Super extends Hyper { }
```

If `Super` and `Hyper` are recompiled but not `Test`, then a `NoSuchMethodError` will result at link time, because the method `hello` is no longer declared in class `Super`.

To preserve binary compatibility, methods should not be deleted; instead, "forwarding methods" should be used. In our example, replacing the declaration of `Super` with:

```
class Super extends Hyper {
    void hello() { super.hello(); }
}
```

then recompiling `Super` and `Hyper` and executing these new binaries with the original binary for `Test`, produces the output:

```
hello from Hyper
```

as might have naively been expected from the previous example.

The `super` keyword can be used to access a method declared in a superclass, bypassing any methods declared in the current class. The expression:

```
super.Identifier
```


is resolved, at compile time, to a method *M* declared in a particular superclass *S*. The method *M* must still be declared in that class at run time or a linkage error will result. If the method *M* is an instance method, then the method *MR* invoked at run time is the method with the same signature as *M* that is a member of the direct superclass of the class containing the expression involving `super`. Thus, if the program:

```
class Hyper {
    void hello() { System.out.println("hello from Hyper"); }
}
class Super extends Hyper { }
class Test extends Super {

    public static void main(String[] args) {
        new Test().hello();
    }

    void hello() {
        super.hello();
    }
}
```

is compiled and executed, it produces the output:

```
hello from Hyper
```

Suppose that a new version of class `Super` is produced:

```
class Super extends Hyper {
    void hello() { System.out.println("hello from Super"); }
}
```

If `Super` and `Hyper` are recompiled but not `Test`, then running the new binaries with the existing binary of `Test` produces the output:

```
hello from Super
```

as you might expect. (A flaw in some early versions of Java caused them to print:

```
hello from Hyper
```

incorrectly.)


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.6 Access to Members and Constructors

Changing the declared access of a member or constructor to permit less access may break compatibility with pre-existing binaries, causing a linkage error to be thrown when these binaries are resolved. Less access is permitted if the access modifier is changed from default access to `private` access; from `protected` access to default or `private` access; or from `public` access to `protected`, default, or `private` access. Changing a member or constructor to permit less access is therefore not recommended for widely distributed classes.

Perhaps surprisingly, Java is defined so that changing a member or constructor to be more accessible does not cause a linkage error when a subclass (already) defines a method to have less access. So, for example, if the package `points` defines the class `Point`:

```
package points;
```

```
public class Point {
    public int x, y;
    protected void print() {
        System.out.println("(" + x + ", " + y + ")");
    }
}
```

used by the `Test` program:

```
class Test extends points.Point {
    protected void print() { System.out.println("Test"); }
    public static void main(String[] args) {
        Test t = new Test();
        t.print();
    }
}
```

then these classes compile and `Test` executes to produce the output:

```
Test
```

If the method `print` in class `Point` is changed to be `public`, and then only the `Point` class is recompiled, and then executed with the previously existing binary for `Test` then no linkage error occurs, even though it is improper, at compile time, for a `public` method to be overridden by a `protected` method (as shown by the fact that the class `Test` could not be recompiled using this new `Point` class unless `print` were changed to be `public`.)

Allowing superclasses to change `protected` methods to be `public` without breaking binaries of preexisting subclasses helps make Java binaries less fragile. The alternative, where such a change would cause a linkage error, would create additional binary incompatibilities with no apparent benefit.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.7 Field Declarations

Adding a field to a class will not break compatibility with any pre-existing binaries that are not recompiled, even in the case where a class could no longer be recompiled because a field access previously referenced a field of a superclass with an incompatible type. The previously compiled class with such a reference will continue to reference the field declared in a superclass. Thus compiling and executing the code:

```
class Hyper { String h = "hyper"; }
class Super extends Hyper { String s = "super"; }
class Test {
    public static void main(String[] args) {
        System.out.println(new Super().h);
    }
}
```

produces the output:

hyper

Changing `Super` to be defined as:

```
class Super extends Hyper {
    String s = "super";
    int h = 0;
}
```

recompiling `Hyper` and `Super`, and executing the resulting new binaries with the old binary of `Test` produces the output:

hyper

The field `h` of `Hyper` is output by the original binary of `main` no matter what type field `h` is declared in `Super`. While this may seem surprising at first, it serves to reduce the number of incompatibilities that occur at run time. (In an ideal world, all source files that needed recompilation would be recompiled whenever any one of them changed, eliminating such surprises. But such a mass recompilation is often impractical or impossible, especially in the Internet. And, as was previously noted, such recompilation would sometimes require further changes to the source code.)

Deleting a field from a class will break compatibility with any pre-existing binaries that reference this field, and a `NoSuchFieldError` will be thrown when such a reference from a pre-existing binary is linked. Only `private` fields may be safely deleted from a widely distributed class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

13.4.8 final Fields and Constants

If a field that was not `final` is changed to be `final`, then it can break compatibility with pre-existing binaries that attempt to assign new values to the field. For example, if the program:

```
class Super { static char s; }

class Test extends Super {
    public static void main(String[] args) {
        s = 'a';
        System.out.println(s);
    }
}
```

is compiled and executed, it produces the output:

a

Suppose that a new version of class `Super` is produced:

```
class Super { static char s; }
```

If `Super` is recompiled but not `Test`, then running the new binary with the existing binary of `Test` results in a `IncompatibleClassChangeError`. (In certain early implementations of Java this example would run without error, because of a flaw in the implementation.)

We call a field that is `static`, `final`, and initialized with a compile-time constant expression a *primitive constant*. Note that all fields in interfaces are implicitly `static` and `final`, and they are often, but not always, constants.

If a field is not a primitive constant, then deleting the keyword `final` or changing the value to which the field is initialized does not break compatibility with existing binaries.

If a field is a primitive constant, then deleting the keyword `final` or changing its value will not break compatibility with pre-existing binaries by causing them not to run, but they will not see any new value for the constant unless they are recompiled. If the example:

```
class Flags { final static boolean debug = true; }

class Test {
    public static void main(String[] args) {
        if (Flags.debug)
            System.out.println("debug is true");
    }
}
```



```
}
```

is compiled and executed, it produces the output:

```
debug is true
```

Suppose that a new version of class `Flags` is produced:

```
class Flags { final static boolean debug = false; }
```

If `Flags` is recompiled but not `Test`, then running the new binary with the existing binary of `Test` produces the output:

```
debug is true
```

because the value of `debug` was a compile-time primitive constant, and could have been used in compiling `Test` without making a reference to the class `Flags`.

This behavior would not change if `Flags` were changed to be an interface, as in the modified example:

```
interface Flags { boolean debug = true; }
class Test {
    public static void main(String[] args) {
        if (Flags.debug)
            System.out.println("debug is true");
    }
}
```

(One reason for requiring inlining of primitive constants is that Java `switch` statements require constants on each `case`, and no two such constant values may be the same. Java checks for duplicate constant values in a `switch` statement at compile time; the `class` file format does not do symbolic linkage of `case` values.)

The best way to avoid problems with "inconstant constants" in widely-distributed code is to declare as primitive constants only values which truly are unlikely ever to change. Many primitive constants in interfaces are small integer values replacing enumerated types, which Java does not support; these small values can be chosen arbitrarily, and should not need to be changed. Other than for true mathematical constants, we recommend that Java code make very sparing use of class variables that are declared `static` and `final`. If the read-only nature of `final` is required, a better choice is to declare a `private static` variable and a suitable accessor method to get its value. Thus we recommend:

```
private static int N;
public static int getN() { return N; }
```

rather than:


```
public static final int N = ...;
```

There is no problem with:

```
public static int N = ...;
```

if `N` need not be read-only. We also recommend, as a general rule, that only truly constant values be declared in interfaces. We note, but do not recommend, that if a field of primitive type of an interface may change, its value may be expressed idiomatically as in:

```
interface Flags {  
    boolean debug = new Boolean(true).booleanValue();  
}
```

insuring that this value is not a constant. Similar idioms exist for the other primitive types.

One other thing to note is that `static final` fields that have constant values (whether of primitive or `String` type) must never appear to have the default initial value for their type ([§4.5.4](#)). This means that all such fields appear to be initialized first during class initialization ([§8.3.2.1](#), [§9.3.1](#), [§12.4.2](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.9 static Fields

If a field that is not declared `private` was not declared `static` and is changed to be declared `static`, or vice versa, then a linkage time error, specifically an `IncompatibleClassChangeError`, will result if the field is used by a preexisting binary which expected a field of the other kind. Such changes are not recommended in code that has been widely distributed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.10 transient Fields

Adding or deleting a `transient` modifier of a field does not break compatibility with pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.11 volatile Fields

If a field that is not declared `private` was not declared `volatile` and is changed to be declared `volatile`, or vice versa, then a linkage time error, specifically an `IncompatibleClassChangeError`, may result if the field is used by a preexisting binary that expected a field of the opposite volatility. Such changes are not recommended in code that has been widely distributed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.12 Method and Constructor Declarations

Adding a method or constructor declaration to a class will not break compatibility with any pre-existing binaries, even in the case where a type could no longer be recompiled because a method invocation previously referenced a method of a superclass with an incompatible type. The previously compiled class with such a reference will continue to reference the method declared in a superclass.

Deleting a method or constructor from a class will break compatibility with any pre-existing binary that referenced this method or constructor; a `NoSuchMethodError` will be thrown when such a reference from a pre-existing binary is linked. Only `private` methods or constructors may be safely deleted from a widely distributed class.

If the source code for a class contains no declared constructors, the Java compiler automatically supplies a constructor with no parameters. Adding one or more constructor declarations to the source code of such a class will prevent this default constructor from being supplied automatically, effectively deleting a constructor, unless one of the new constructors also has no parameters, thus replacing the default constructor. The automatically supplied constructor with no parameters is given the same access modifier as the class of its declaration, so any replacement should have as much or more access if compatibility with pre-existing binaries is to be preserved.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.13 Method and Constructor Parameters

Changing the name of a formal parameter of a method or constructor does not impact pre-existing binaries. Changing the name of a method, the type of a formal parameter to a method or constructor, or adding a parameter to or deleting a parameter from a method or constructor declaration creates a method or constructor with a new signature, and has the combined effect of deleting the method or constructor with the old signature and adding a method or constructor with the new signature (see §13.4.12).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.14 Method Result Type

Changing the result type of a method, replacing a result type with `void`, or replacing `void` with a result type has the combined effect of deleting the old method or constructor and adding a new method or constructor with the new result type or newly `void` result (see [§13.4.12](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.15 abstract Methods

Changing a method that is declared `abstract` to no longer be declared `abstract` does not break compatibility with pre-existing binaries.

Changing a method that is not declared `abstract` to be declared `abstract` will break compatibility with pre-existing binaries that previously invoked the method, causing an `AbstractMethodError`. If the example program:

```
class Super { void out() { System.out.println("Out"); } }

class Test extends Super {
    public static void main(String[] args) {
        Test t = new Test();
        System.out.println("Way ");
        t.out();
    }
}
```

is compiled and executed, it produces the output:

```
Way
Out
```

Suppose that a new version of class `Super` is produced:

```
abstract class Super {
    abstract void out();
}
```

If `Super` is recompiled but not `Test`, then running the new binary with the existing binary of `Test` results in a `AbstractMethodError`, because class `Test` has no implementation of the method `out`, and is therefore is (or should be) `abstract`. (An early version of Java incorrectly produced the output:

```
Way
```

before encountering an `AbstractMethodError` while invoking the method `out`, incorrectly allowing the class `Test` to be prepared even though it has an `abstract` method and is not declared `abstract`.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.16 final Methods

Changing an instance method that is not `final` to be `final` may break compatibility with existing binaries that depend on the ability to override the method. If the test program:

```
class Super { void out() { System.out.println("out"); } }
class Test extends Super {

    public static void main(String[] args) {
        Test t = new Test();
        t.out();
    }
    void out() { super.out(); }
}
```

is compiled and executed, it produces the output:

```
out
```

Suppose that a new version of class `Super` is produced:

```
class Super { final void out() { System.out.println("!"); } }
```

If `Super` is recompiled but not `Test`, then running the new binary with the existing binary of `Test` results in a `VerifyError` because the class `Test` improperly tries to override the instance method `out`.

Changing a class (static) method that is not `final` to be `final` does not break compatibility with existing binaries, because the class of the actual method to be invoked is resolved at compile time.

Removing the `final` modifier from a method does not break compatibility with pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.17 native Methods

Adding or deleting a `native` modifier of a method does not break compatibility with pre-existing binaries.

The impact of changes to Java types on preexisting `native` methods that are not recompiled is beyond the scope of this specification and should be provided with the description of an implementation of Java. Implementations are encouraged, but not required, to implement `native` methods in a way that limits such impact.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.18 static Methods

If a method that is not declared `private` was declared `static` (that is, a class method) and is changed to not be declared `static` (that is, to an instance method), or vice versa, then compatibility with pre-existing binaries may be broken, resulting in a linkage time error, namely an `IncompatibleClassChangeError`, if these methods are used by the pre-existing binaries. Such changes are not recommended in code that has been widely distributed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.19 synchronized Methods

Adding or deleting a `synchronized` modifier of a method does not break compatibility with existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.20 Method and Constructor Throws

Changes to the `throws` clause of methods or constructors do not break compatibility with existing binaries; these clauses are checked only at compile time.

We are considering whether a future version of the Java language should require more rigorous checking of `throws` clauses when classes are verified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.21 Method and Constructor Body

Changes to the body of a method or constructor do not break compatibility with pre-existing binaries.

We note that a compiler cannot inline expand a method at compile time unless, for example, either:

- the method is `private` to its class
- an entire package is guaranteed to be kept together and the method is accessible only within that package
- a set of Java code is being compiled to a special binary format where the specified method is available only within a binary or set of binaries which are being kept together.

The keyword `final` on a method does not mean that the method can be safely inlined; it only means that the method cannot be overridden. Unless the compiler has extraordinary knowledge, it is still possible that a new version of that method will be provided at link time.

In general we suggest that Java implementations use late-bound (run-time) code generation and optimization.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.22 Method and Constructor Overloading

Adding new methods that overload existing method names does not break compatibility with pre-existing binaries. The method signature to be used for each method invocation was determined when these existing binaries were compiled; therefore newly added methods will not be used, even if their signatures are both applicable and more specific than the method signature originally chosen.

While adding a new overloaded method or constructor may cause a compile-time error the next time a class or interface is compiled because there is no method or constructor that is most specific ([§15.11.2.2](#)), no such error occurs when a Java program is executed, because no overload resolution is done at execution time.

If the example program:

```
class Super {
    static void out(float f) { System.out.println("float"); }
}

class Test {
    public static void main(String[] args) {
        Super.out(2);
    }
}
```

is compiled and executed, it produces the output:

```
float
```

Suppose that a new version of class `Super` is produced:

```
class Super {
    static void out(float f) { System.out.println("float"); }
    static void out(int i) { System.out.println("int"); }
}
```

If `Super` is recompiled but not `Test`, then running the new binary with the existing binary of `Test` still produces the output:

```
float
```

However, if `Test` is then recompiled, using this new `Super`, the output is then:

```
int
```

as might have been naively expected in the previous case.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.23 Method Overriding

If an instance method is added to a subclass and it overrides a method in a superclass, then the subclass method will be found by method invocations in pre-existing binaries, and these binaries are not impacted. If a class method is added to a class, then this method will not be found, because the invocation of a class method is resolved at compile time to use the fully qualified name of the class where the method is declared. Thus if the example:

```
class Hyper {
    void hello() { System.out.print("Hello, "); }
    static void world() { System.out.println("world!"); }
}
class Super extends Hyper { }

class Test {
    public static void main(String[] args) {
        Super s = new Super();
        s.hello();
        s.world();
    }
}
```

is compiled and executed, it produces the output:

```
Hello, world!
```

Suppose that a new version of class `Super` is produced:

```
class Super extends Hyper {
    void hello() { System.out.print("Goodbye, cruel "); }
    static void world() { System.out.println("earth!"); }
}
```

If `Super` is recompiled but not `Hyper` or `Test`, then running the new binary with the existing binaries for `Hyper` and `Test` will produce the output:

```
Goodbye, cruel world!
```

This example demonstrates that the invocation in:

```
s.world();
```

in the method `main` is resolved, at compile time, to a symbolic reference to the class containing the class method `world`, as though it had been written:


```
Hyper.world();
```

This is why the `world` method of `Hyper` rather than `Super` is invoked in this example. Of course, recompiling all the classes to produce new binaries will allow the output:

```
Goodbye, cruel earth!
```

to be produced.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.4.24 Static Initializers

Adding, deleting, or changing a static initializer ([§8.5](#)) of a class does not impact pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5 Evolution of Interfaces

This section describes the impact of changes to the declaration of an interface and its members on pre-existing binaries.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5.1 public Interfaces

Changing an interface that is not declared `public` to be declared `public` does not break compatibility with pre-existing binaries.

If an interface that is declared `public` is changed to not be declared `public`, then an `IllegalAccessError` is thrown if a pre-existing binary is linked that needs but no longer has access to the interface type, so such a change is not recommended for widely distributed interfaces.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5.2 Superinterfaces

Changes to the interface hierarchy cause errors in the same way that changes to the class hierarchy do, as described in §13.4.4. In particular, changes that result in any previous superinterface of a class no longer being a superinterface can break compatibility with pre-existing binaries, resulting in a `VerifyError`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5.3 The Interface Members

Adding a member to an interface does not break compatibility with pre-existing binaries.

Deleting a member from an interface may cause linkage errors in pre-existing binaries. If the example program:

```
interface I { void hello(); }
class Test implements I {

    public static void main(String[] args) {
        I anI = new Test();
        anI.hello();
    }
    public void hello() { System.out.println("hello"); }
}
```

is compiled and executed, it produces the output:

```
hello
```

Suppose that a new version of interface `I` is compiled:

```
interface I { }
```

If `I` is recompiled but not `Test`, then running the new binary with the existing binary for `Test` will result in a `NoSuchMethodError`. (In some early implementations of Java this program still executed; the fact that the method `hello` no longer exists in interface `I` was not correctly detected.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5.4 Field Declarations

The considerations for changing field declarations in interfaces are the same as those for `static final` fields in classes, as described in [§13.4.7](#) and [§13.4.8](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


13.5.5 Abstract Method Declarations

The considerations for changing abstract method declarations in interfaces are the same as those for `abstract` methods in classes, as described in [§13.4.13](#), [§13.4.14](#), [§13.4.20](#), and [§13.4.22](#).

Lo! keen-eyed, towering Science! . . .

Yet again, lo! the Soul--above all science . . .

For it, the partial to the permanent flowing,

For it, the Real to the Ideal tends.

For it, the mystic evolution . . .

--Walt Whitman, *Song of the Universal* (1874)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Blocks and Statements

He was not merely a chip of the old block, but the old block itself.

--Edmund Burke, On Pitt's First Speech

The sequence of execution of a Java program is controlled by *statements*, which are executed for their effect and do not have values.

Some statements *contain* other statements as part of their structure; such other statements are substatements of the statement. We say that statement *S* *immediately contains* statement *U* if there is no statement *T* different from *S* and *U* such that *S* contains *T* and *T* contains *U*. In the same manner, some statements contain expressions (§15) as part of their structure.

The first section of this chapter discusses the distinction between normal and abrupt completion of statements (§14.1). Most of the remaining sections explain the various kinds of statements, describing in detail both their normal behavior and any special treatment of abrupt completion.

Blocks are explained first (§14.2), because they can appear in certain places where other kinds of statements are not allowed, and because one other kind of statement, a local variable declaration statement (§14.3), must be immediately contained within a block.

Next a grammatical maneuver is explained that sidesteps the familiar "dangling `else`" problem (§14.4).

Statements that will be familiar to C and C++ programmers are the empty (§14.5), labeled (§14.6), expression (§14.7), `if` (§14.8), `switch` (§14.9), `while` (§14.10), `do` (§14.11), `for` (§14.12), `break` (§14.13), `continue` (§14.14), and `return` (§14.15) statements.

Unlike C and C++, Java has no `goto` statement. However, the `break` and `continue` statements are extended in Java to allow them to mention statement labels.

The Java statements that are not in the C language are the `throw` (§14.16), `synchronized` (§14.17), and `try` (§14.18) statements.

The last section (§14.19) of this chapter addresses the requirement that every statement be *reachable* in a certain technical sense.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.1 Normal and Abrupt Completion of Statements

Poirot's abrupt departure had intrigued us all greatly.

--Agatha Christie, *The Mysterious Affair at Styles* (1920), Chapter 12

Every statement has a normal mode of execution in which certain computational steps are carried out. The following sections describe the normal mode of execution for each kind of statement. If all the steps are carried out as described, with no indication of abrupt completion, the statement is said to *complete normally*. However, certain events may prevent a statement from completing normally:

- The `break` (§14.13), `continue` (§14.14), and `return` (§14.15) statements cause a transfer of control that may prevent normal completion of statements that contain them.
- Evaluation of certain Java expressions may throw exceptions from the Java Virtual Machine; these expressions are summarized in §15.5. An explicit `throw` (§14.16) statement also results in an exception. An exception causes a transfer of control that may prevent normal completion of statements.

If such an event occurs, then execution of one or more statements may be terminated before all steps of their normal mode of execution have completed; such statements are said to *complete abruptly*. An abrupt completion always has an associated *reason*, which is one of the following:

- A `break` with no label
- A `break` with a given label
- A `continue` with no label
- A `continue` with a given label
- A `return` with no value
- A `return` with a given value
- A `throw` with a given value, including exceptions thrown by the Java Virtual Machine

The terms "complete normally" and "complete abruptly" also apply to the evaluation of expressions (§15.5). The only reason an expression can complete abruptly is that an exception is thrown, because of either a `throw` with a given value (§14.16) or a run-time exception or error (§11, §15.5).

If a statement evaluates an expression, abrupt completion of the expression always causes the immediate abrupt completion of the statement, with the same reason. All succeeding steps in the normal mode of execution are not performed.

Unless otherwise specified in this chapter, abrupt completion of a substatement causes the immediate abrupt completion of the statement itself, with the same reason, and all succeeding steps in the normal mode of execution of the statement are not performed.

Unless otherwise specified, a statement completes normally if all expressions it evaluates and all substatements it executes complete normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.2 Blocks

A *block* is a sequence of statements and local variable declaration statements within braces.

Block:

{ BlockStatementsopt }

BlockStatements:

BlockStatement

BlockStatements BlockStatement

BlockStatement:

LocalVariableDeclarationStatement

Statement

A block is executed by executing each of the local variable declaration statements and other statements in order from first to last (left to right). If all of these block statements complete normally, then the block completes normally. If any of these block statements complete abruptly for any reason, then the block completes abruptly for the same reason.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.3 Local Variable Declaration Statements

A *local variable declaration statement* declares one or more local variable names.

LocalVariableDeclarationStatement:

LocalVariableDeclaration ;

LocalVariableDeclaration:

Type VariableDeclarators

The following are repeated from [§8.3](#) to make the presentation here clearer:

VariableDeclarators:

VariableDeclarator

VariableDeclarators , VariableDeclarator

VariableDeclarator:

VariableDeclaratorId

VariableDeclaratorId = VariableInitializer

VariableDeclaratorId:

Identifier

VariableDeclaratorId []

VariableInitializer:

Expression

ArrayInitializer

Every local variable declaration statement is immediately contained by a block. Local variable declaration statements may be intermixed freely with other kinds of statements in the block.

A local variable declaration can also appear in the header of a `for` statement ([§14.12](#)). In this case it is executed in the same manner as if it were part of a local variable declaration statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.3.1 Local Variable Declarators and Types

Each *declarator* in a local variable declaration declares one local variable, whose name is the *Identifier* that appears in the declarator.

The type of the variable is denoted by the *Type* that appears at the start of the local variable declaration, followed by any bracket pairs that follow the *Identifier* in the declarator. Thus, the local variable declaration:

```
int a, b[], c[][];
```

is equivalent to the series of declarations:

```
int a;  
int[] b;  
int[][] c;
```

Brackets are allowed in declarators as a nod to the tradition of C and C++. The general rule, however, also means that the local variable declaration:

```
float[][] f[], g[][][], h[];    // Yechh!
```

is equivalent to the series of declarations:

```
float[][][] f;  
float[][][][] g;  
float[][][] h;
```

We do not recommend such "mixed notation" for array declarations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.3.2 Scope of Local Variable Declarations

The scope of a local variable declared in a block is the rest of the block, including its own initializer. The name of the local variable parameter may not be redeclared as a local variable or exception parameter within its scope, or a compile-time error occurs; that is, hiding the name of a local variable is not permitted.

A local variable cannot be referred to using a qualified name (\$6.6), only a simple name.

The example:

```
class Test {
    static int x;
    public static void main(String[] args) {
        int x = x;
    }
}
```

causes a compile-time error because the initialization of `x` is within the scope of the declaration of `x` as a local variable, and the local `x` does not yet have a value and cannot be used.

The following program does compile:

```
class Test {
    static int x;
    public static void main(String[] args) {
        int x = (x=2)*2;
        System.out.println(x);
    }
}
```

because the local variable `x` is definitely assigned (\$16) before it is used. It prints:

4

Here is another example:

```
class Test {
    public static void main(String[] args) {
        System.out.print("2+1=");
        int two = 2, three = two + 1;
        System.out.println(three);
    }
}
```

which compiles correctly and produces the output:

2+1=3

The initializer for `three` can correctly refer to the variable `two` declared in an earlier declarator, and the method invocation in the next line can correctly refer to the variable `three` declared earlier in the block.

The scope of a local variable declared in a `for` statement is the rest of the `for` statement, including its own initializer.

If a declaration of an identifier as a local variable appears within the scope of a parameter or local variable of the same name, a compile-time error occurs. Thus the following example does not compile:

```
class Test {
    public static void main(String[] args) {
        int i;
        for (int i = 0; i < 10; i++)
            System.out.println(i);
    }
}
```

This restriction helps to detect some otherwise very obscure bugs. (A similar restriction on hiding of members by local variables was judged impractical, because the addition of a member in a superclass could cause subclasses to have to rename local variables.)

On the other hand, local variables with the same name may be declared in two separate blocks or `for` statements neither of which contains the other. Thus:

```
class Test {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++)
            System.out.print(i + " ");
        for (int i = 10; i > 0; i--)
            System.out.print(i + " ");
        System.out.println();
    }
}
```

compiles without error and, when executed, produces the output:

```
0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

14.3.3 Hiding of Names by Local Variables

If a name declared as a local variable is already declared as a field or type name, then that outer declaration is hidden throughout the scope of the local variable. The field or type name can almost always (§6.8) still be accessed using an appropriately qualified name. For example, the keyword `this` can be used to access a hidden field `x`, using the form `this.x`. Indeed, this idiom typically appears in constructors (§8.6):

```
class Pair {
    Object first, second;
    public Pair(Object first, Object second) {
        this.first = first;
        this.second = second;
    }
}
```

In this example, the constructor takes parameters having the same names as the fields to be initialized. This is simpler than having to invent different names for the parameters and is not too confusing in this stylized context. In general, however, it is considered poor style to have local variables with the same names as fields.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.3.4 Execution of Local Variable Declarations

A local variable declaration statement is an executable statement. Every time it is executed, the declarators are processed in order from left to right. If a declarator has an initialization expression, the expression is evaluated and its value is assigned to the variable. If a declarator does not have an initialization expression, then a Java compiler must prove, using exactly the algorithm given in §16, that every reference to the variable is necessarily preceded by execution of an assignment to the variable. If this is not the case, then a compile-time error occurs.

Each initialization (except the first) is executed only if the evaluation of the preceding initialization expression completes normally. Execution of the local variable declaration completes normally only if evaluation of the last initialization expression completes normally; if the local variable declaration contains no initialization expressions, then executing it always completes normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.4 Statements

There are many kinds of statements in the Java language. Most correspond to statements in the C and C++ languages, but some are unique to Java.

As in C and C++, the Java `if` statement suffers from the so-called "dangling `else` problem," illustrated by this misleadingly formatted example:

```
if (door.isOpen())
    if (resident.isVisible())
        resident.greet("Hello!");
else door.bell.ring();                                // A
"dangling else"
```

The problem is that both the outer `if` statement and the inner `if` statement might conceivably own the `else` clause. In this example, one might surmise that the programmer intended the `else` clause to belong to the outer `if` statement. The Java language, like C and C++ and many languages before them, arbitrarily decree that an `else` clause belongs to the innermost `if` to which it might possibly belong. This rule is captured by the following grammar:

Statement:

StatementWithoutTrailingSubstatement

LabeledStatement

IfThenStatement

IfThenElseStatement

WhileStatement

ForStatement

StatementNoShortIf:

StatementWithoutTrailingSubstatement

LabeledStatementNoShortIf

IfThenElseStatementNoShortIf

WhileStatementNoShortIf

ForStatementNoShortIf

StatementWithoutTrailingSubstatement:

Block

EmptyStatement

ExpressionStatement

SwitchStatement

DoStatement

BreakStatement

ContinueStatement

ReturnStatement

SynchronizedStatement

ThrowStatement

TryStatement

The following are repeated from §14.8 to make the presentation here clearer:

IfThenStatement:

if (Expression) Statement

IfThenElseStatement:

if (Expression) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:

if (Expression) StatementNoShortIf else StatementNoShortIf

Statements are thus grammatically divided into two categories: those that might end in an `if` statement that has no `else` clause (a "short `if` statement") and those that definitely do not. Only statements that definitely do not end in a short `if` statement may appear as an immediate substatement before the keyword `else` in an `if` statement that does have an `else` clause. This simple rule prevents the "dangling `else`" problem. The execution behavior of a statement with the "no short `if`" restriction is identical to the execution behavior of the same kind of statement without the "no short `if`" restriction; the distinction is drawn purely to resolve the syntactic difficulty.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

14.5 The Empty Statement

An *empty statement* does nothing.

EmptyStatement:

;

Execution of an empty statement always completes normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.6 Labeled Statements

Statements may have *label* prefixes.

LabeledStatement:

Identifier : *Statement*

LabeledStatementNoShortIf:

Identifier : *StatementNoShortIf*

The *Identifier* is declared to be the label of the immediately contained *Statement*.

Unlike C and C++, the Java language has no `goto` statement; identifier statement labels are used with `break` (§14.13) or `continue` (§14.14) statements appearing anywhere within the labeled statement.

A statement labeled by an identifier must not appear anywhere within another statement labeled by the same identifier, or a compile-time error will occur. Two statements can be labeled by the same identifier only if neither statement contains the other.

There is no restriction against using the same identifier as a label and as the name of a package, class, interface, method, field, parameter, or local variable. Use of an identifier to label a statement does not hide a package, class, interface, method, field, parameter, or local variable with the same name. Use of an identifier as a local variable or as the parameter of an exception handler (§14.18) does not hide a statement label with the same name.

A labeled statement is executed by executing the immediately contained *Statement*. If the statement is labeled by an *Identifier* and the contained *Statement* completes abruptly because of a `break` with the same *Identifier*, then the labeled statement completes normally. In all other cases of abrupt completion of the *Statement*, the labeled statement completes abruptly for the same reason.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.7 Expression Statements

Certain kinds of expressions may be used as statements by following them with semicolons:

ExpressionStatement:

StatementExpression ;

StatementExpression:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

An *expression statement* is executed by evaluating the expression; if the expression has a value, the value is discarded. Execution of the expression statement completes normally if and only if evaluation of the expression completes normally.

Unlike C and C++, the Java language allows only certain forms of expressions to be used as expression statements. Note that Java does not allow a "cast to `void`"-`void` is not a type in Java-so the traditional C trick of writing an expression statement such as:

```
(void) ... ;           // This idiom belongs to C, not to Java!
```

does not work in Java. On the other hand, Java allows all the most useful kinds of expressions in expressions statements, and Java does not require a method invocation used as an expression statement to invoke a `void` method, so such a trick is almost never needed. If a trick is needed, either an assignment statement ([§15.25](#)) or a local variable declaration statement ([§14.3](#)) can be used instead.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.8 The if Statement

The `if` statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

IfThenStatement:

if (Expression) Statement

IfThenElseStatement:

if (Expression) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:

if (Expression) StatementNoShortIf else StatementNoShortIf

The *Expression* must have type `boolean`, or a compile-time error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.8.1 The if-then Statement

I took an early opportunity of testing that statement . . .

--Agatha Christie, *The Mysterious Affair at Styles* (1920), Chapter 12

An `if-then` statement is executed by first evaluating the *Expression*. If evaluation of the *Expression* completes abruptly for some reason, the `if-then` statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the contained *Statement* is executed; the `if-then` statement completes normally only if execution of the *Statement* completes normally.
- If the value is `false`, no further action is taken and the `if-then` statement completes normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.8.2 The if-then-else Statement

An `if-then-else` statement is executed by first evaluating the *Expression*. If evaluation of the *Expression* completes abruptly for some reason, then the `if-then-else` statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the first contained *Statement* (the one before the `else` keyword) is executed; the `if-then-else` statement completes normally only if execution of that statement completes normally.
- If the value is `false`, then the second contained *Statement* (the one after the `else` keyword) is executed; the `if-then-else` statement completes normally only if execution of that statement completes normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.9 The switch Statement

The `switch` statement transfers control to one of several statements depending on the value of an expression.

SwitchStatement:

```
switch ( Expression ) SwitchBlock
```

SwitchBlock:

```
{ SwitchBlockStatementGroupsopt SwitchLabelsopt }
```

SwitchBlockStatementGroups:

```
SwitchBlockStatementGroup
```

```
SwitchBlockStatementGroups SwitchBlockStatementGroup
```

SwitchBlockStatementGroup:

```
SwitchLabels BlockStatements
```

SwitchLabels:

```
SwitchLabel
```

```
SwitchLabels SwitchLabel
```

SwitchLabel:

```
case ConstantExpression :
```

```
default :
```

The type of the *Expression* must be `char`, `byte`, `short`, or `int`, or a compile-time error occurs.

The body of a `switch` statement must be a block. Any statement immediately contained by the block may be labeled with one or more `case` or `default` labels. These labels are said to be *associated* with the `switch` statement, as are the values of the constant expressions ([§15.27](#)) in the `case` labels.

All of the following must be true, or a compile-time error will result:

- Every `case` constant expression associated with a `switch` statement must be assignable ([§5.2](#)) to the type of the `switch Expression`.
- No two of the `case` constant expressions associated with a `switch` statement may have the same value.
- At most one `default` label may be associated with the same `switch` statement.

In C and C++ the body of a `switch` statement can be a statement and statements with `case` labels do not have to be immediately contained by that statement. Consider the simple loop:


```
for (i = 0; i < n; ++i) foo();
```

where `n` is known to be positive. A trick known as *Duff's device* can be used in C or C++ to unroll the loop, but this is not valid Java code:

```
int q = (n+7)/8;
switch (n%8) {
case 0:           do {           foo(); // Great C hack, Tom,
case 7:           foo(); // but it's not valid in Java.
case 6:           foo();
case 5:           foo();
case 4:           foo();
case 3:           foo();
case 2:           foo();
case 1:           foo();
                  } while (--q >= 0);
}
```

Fortunately, this trick does not seem to be widely known or used. Moreover, it is less needed nowadays; this sort of code transformation is properly in the province of state-of-the-art optimizing compilers.

When the `switch` statement is executed, first the *Expression* is evaluated. If evaluation of the *Expression* completes abruptly for some reason, the `switch` statement completes abruptly for the same reason. Otherwise, execution continues by comparing the value of the *Expression* with each `case` constant. Then there is a choice:

- If one of the `case` constants is equal to the value of the expression, then we say that the `case` matches, and all statements after the matching `case` label in the switch block, if any, are executed in sequence. If all these statements complete normally, or if there are no statements after the matching `case` label, then the entire `switch` statement completes normally.
- If no `case` matches but there is a `default` label, then all statements after the matching `default` label in the switch block, if any, are executed in sequence. If all these statements complete normally, or if there are no statements after the `default` label, then the entire `switch` statement completes normally.
- If no `case` matches and there is no `default` label, then no further action is taken and the `switch` statement completes normally.

If any statement immediately contained by the *Block* body of the `switch` statement completes abruptly, it is handled as follows:

- If execution of the *Statement* completes abruptly because of a `break` with no label, no further action is taken and the `switch` statement completes normally.
- If execution of the *Statement* completes abruptly for any other reason, the `switch` statement completes abruptly for the same reason. The case of abrupt completion because of a `break` with a label is handled by the general rule for labeled statements ([§14.6](#)).

As in C and C++, execution of statements in a switch block "falls through labels" in Java. For example, the program:


```

class Toomany {

    static void howMany(int k) {
        switch (k) {
            case 1:          System.out.print("one ");
            case 2:          System.out.print("too ");
            case 3:          System.out.println("many");
        }
    }

    public static void main(String[] args) {
        howMany(3);
        howMany(2);
        howMany(1);
    }
}

```

contains a switch block in which the code for each case falls through into the code for the next case. As a result, the program prints:

```

many
too many
one too many

```

If code is not to fall through case to case in this manner, then `break` statements should be used, as in this example:

```

class Twomany {

    static void howMany(int k) {
        switch (k) {
            case 1:          System.out.println("one");
                           break; // exit the switch
            case 2:          System.out.println("two");
                           break; // exit the switch
            case 3:          System.out.println("many");
                           break; // not needed, but good style
        }
    }

    public static void main(String[] args) {
        howMany(1);
        howMany(2);
        howMany(3);
    }
}

```


This program prints:

one
two
many

{ewl msdncd.dll, ewcright, /c"Microsoft"}

14.10 The while Statement

The `while` statement executes an *Expression* and a *Statement* repeatedly until the value of the *Expression* is `false`.

WhileStatement:

```
while ( Expression ) Statement
```

WhileStatementNoShortIf:

```
while ( Expression ) StatementNoShortIf
```

The *Expression* must have type `boolean`, or a compile-time error occurs.

A `while` statement is executed by first evaluating the *Expression*. If evaluation of the *Expression* completes abruptly for some reason, the `while` statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the contained *Statement* is executed. Then there is a choice:
 - If execution of the *Statement* completes normally, then the entire `while` statement is executed again, beginning by re-evaluating the *Expression*.
 - If execution of the *Statement* completes abruptly, see §14.10.1 below.
- If the value of the *Expression* is `false`, no further action is taken and the `while` statement completes normally.

If the value of the *Expression* is `false` the first time it is evaluated, then the *Statement* is not executed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.10.1 Abrupt Completion

Abrupt completion of the contained *Statement* is handled in the following manner:

- If execution of the *Statement* completes abruptly because of a `break` with no label, no further action is taken and the `while` statement completes normally.
 - If execution of the *Statement* completes abruptly because of a `continue` with no label, then the entire `while` statement is executed again.
 - If execution of the *Statement* completes abruptly because of a `continue` with label *L*, then there is a choice:
 - If the `while` statement has label *L*, then the entire `while` statement is executed again.
 - If the `while` statement does not have label *L*, the `while` statement completes abruptly because of a `continue` with label *L*.
 - If execution of the *Statement* completes abruptly for any other reason, the `while` statement completes abruptly for the same reason. Note that the case of abrupt completion because of a `break` with a label is handled by the general rule for labeled statements ([§14.6](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.11 The do Statement

The `do` statement executes a *Statement* and an *Expression* repeatedly until the value of the *Expression* is `false`.

DoStatement:

```
do Statement while ( Expression ) ;
```

The *Expression* must have type `boolean`, or a compile-time error occurs.

A `do` statement is executed by first executing the *Statement*. Then there is a choice:

- If execution of the *Statement* completes normally, then the *Expression* is evaluated. If evaluation of the *Expression* completes abruptly for some reason, the `do` statement completes abruptly for the same reason. Otherwise, there is a choice based on the resulting value:
 - If the value is `true`, then the entire `do` statement is executed again.
 - If the value is `false`, no further action is taken and the `do` statement completes normally.
- If execution of the *Statement* completes abruptly, see [§14.11.1](#) below.

Executing a `do` statement always executes the contained *Statement* at least once.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.11.1 Abrupt Completion

Abrupt completion of the contained *Statement* is handled in the following manner:

- If execution of the *Statement* completes abruptly because of a `break` with no label, then no further action is taken and the `do` statement completes normally.
- If execution of the *Statement* completes abruptly because of a `continue` with no label, then the *Expression* is evaluated. Then there is a choice based on the resulting value:
 - If the value is `true`, then the entire `do` statement is executed again.
 - If the value is `false`, no further action is taken and the `do` statement completes normally.
- If execution of the *Statement* completes abruptly because of a `continue` with label *L*, then there is a choice:
 - If the `do` statement has label *L*, then the *Expression* is evaluated. Then there is a choice:
 - If the value of the *Expression* is `true`, then the entire `do` statement is executed again.
 - If the value of the *Expression* is `false`, no further action is taken and the `do` statement completes normally.
 - If the `do` statement does not have label *L*, the `do` statement completes abruptly because of a `continue` with label *L*.
- If execution of the *Statement* completes abruptly for any other reason, the `do` statement completes abruptly for the same reason. The case of abrupt completion because of a `break` with a label is handled by the general rule ([§14.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

14.11.2 Example

The following code is one possible implementation of the `toHexString` method ([§20.7.14](#)) of class `Integer`:

```
public static String toHexString(int i) {
    StringBuffer buf = new StringBuffer(8);
    do {
        buf.append(Character.forDigit(i & 0xF, 16));
        i >>= 4;
    } while (i != 0);
    return buf.reverse().toString();
}
```

Because at least one digit must be generated, the `do` statement is an appropriate control structure.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.12 The for Statement

The `for` statement executes some initialization code, then executes an *Expression*, a *Statement*, and some update code repeatedly until the value of the *Expression* is `false`.

ForStatement:

```
for ( ForInitopt ; Expressionopt ; ForUpdateopt )  
    Statement
```

ForStatementNoShortIf:

```
for ( ForInitopt ; Expressionopt ; ForUpdateopt )  
    StatementNoShortIf
```

ForInit:

```
StatementExpressionList  
LocalVariableDeclaration
```

ForUpdate:

```
StatementExpressionList
```

StatementExpressionList:

```
StatementExpression  
StatementExpressionList , StatementExpression
```

The *Expression* must have type `boolean`, or a compile-time error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.12.1 Initialization

A `for` statement is executed by first executing the *ForInit* code:

- If the *ForInit* code is a list of statement expressions (§14.7), the expressions are evaluated in sequence from left to right; their values, if any, are discarded. If evaluation of any expression completes abruptly for some reason, the `for` statement completes abruptly for the same reason; any *ForInit* statement expressions to the right of the one that completed abruptly are not evaluated.
- If the *ForInit* code is a local variable declaration, it is executed as if it were a local variable declaration statement (§14.3) appearing in a block. In this case, the scope of a declared local variable is its own initializer and any further declarators in the *ForInit* part, plus the *Expression*, *ForUpdate*, and contained *Statement* of the `for` statement. If execution of the local variable declaration completes abruptly for any reason, the `for` statement completes abruptly for the same reason.
- If the *ForInit* part is not present, no action is taken.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.12.2 Iteration

Next, a `for` iteration step is performed, as follows:

- If the *Expression* is present, it is evaluated, and if evaluation of the *Expression* completes abruptly, the `for` statement completes abruptly for the same reason. Otherwise, there is then a choice based on the presence or absence of the *Expression* and the resulting value if the *Expression* is present:
 - If the *Expression* is not present, or it is present and the value resulting from its evaluation is `true`, then the contained *Statement* is executed. Then there is a choice:
 - If execution of the *Statement* completes normally, then the following two steps are performed in sequence:
 - **First, if the *ForUpdate* part is present, the expressions are evaluated in sequence from left to right; their values, if any, are discarded. If evaluation of any expression completes abruptly for some reason, the `for` statement completes abruptly for the same reason; any *ForUpdate* statement expressions to the right of the one that completed abruptly are not evaluated. If the *ForUpdate* part is not present, no action is taken.**
 - **Second, another `for` iteration step is performed.**
 - If execution of the *Statement* completes abruptly, see [§14.12.3](#) below.
 - If the *Expression* is present and the value resulting from its evaluation is `false`, no further action is taken and the `for` statement completes normally.
-

If the value of the *Expression* is `false` the first time it is evaluated, then the *Statement* is not executed.

If the *Expression* is not present, then the only way a `for` statement can complete normally is by use of a `break` statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.12.3 Abrupt Completion

Abrupt completion of the contained *Statement* is handled in the following manner:

- If execution of the *Statement* completes abruptly because of a `break` with no label, no further action is taken and the `for` statement completes normally.
- If execution of the *Statement* completes abruptly because of a `continue` with no label, then the following two steps are performed in sequence:
 - First, if the *ForUpdate* part is present, the expressions are evaluated in sequence from left to right; their values, if any, are discarded. If the *ForUpdate* part is not present, no action is taken.
 - Second, another `for` iteration step is performed.
- If execution of the *Statement* completes abruptly because of a `continue` with label *L*, then there is a choice:
 - If the `for` statement has label *L*, then the following two steps are performed in sequence:
 - First, if the *ForUpdate* part is present, the expressions are evaluated in sequence from left to right; their values, if any, are discarded. If the *ForUpdate* is not present, no action is taken.
 - Second, another `for` iteration step is performed.
 - If the `for` statement does not have label *L*, the `for` statement completes abruptly because of a `continue` with label *L*.
- If execution of the *Statement* completes abruptly for any other reason, the `for` statement completes abruptly for the same reason. Note that the case of abrupt completion because of a `break` with a label is handled by the general rule for labeled statements ([§14.6](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.13 The break Statement

A break statement transfers control out of an enclosing statement.

BreakStatement:

```
break Identifieropt ;
```

A `break` statement with no label attempts to transfer control to the innermost enclosing `switch`, `while`, `do`, or `for` statement; this statement, which is called the *break target*, then immediately completes normally. To be precise, a `break` statement with no label always completes abruptly, the reason being a `break` with no label. If no `switch`, `while`, `do`, or `for` statement encloses the `break` statement, a compile-time error occurs.

A `break` statement with label *Identifier* attempts to transfer control to the enclosing labeled statement (§14.6) that has the same *Identifier* as its label; this statement, which is called the *break target*, then immediately completes normally. In this case, the `break` target need not be a `while`, `do`, `for`, or `switch` statement. To be precise, a `break` statement with label *Identifier* always completes abruptly, the reason being a `break` with label *Identifier*. If no labeled statement with *Identifier* as its label encloses the `break` statement, a compile-time error occurs.

It can be seen, then, that a `break` statement always completes abruptly.

The preceding descriptions say "attempts to transfer control" rather than just "transfers control" because if there are any `try` statements (§14.18) within the break target whose `try` blocks contain the `break` statement, then any `finally` clauses of those `try` statements are executed, in order, innermost to outermost, before control is transferred to the break target. Abrupt completion of a `finally` clause can disrupt the transfer of control initiated by a `break` statement.

In the following example, a mathematical graph is represented by an array of arrays. A graph consists of a set of nodes and a set of edges; each edge is an arrow that points from some node to some other node, or from a node to itself. In this example it is assumed that there are no redundant edges; that is, for any two nodes *P* and *Q*, where *Q* may be the same as *P*, there is at most one edge from *P* to *Q*. Nodes are represented by integers, and there is an edge from node *i* to node *j* if and only if `edges[i][j]` is non-zero for every *i* and *j* for which the array reference `edges[i][j]` does not throw an `IndexOutOfBoundsException`.

The task of the method `loseEdges`, given integers *i* and *j*, is to construct a new graph by copying a given graph but omitting the edge from node *i* to node *j*, if any, and the edge from node *j* to node *i*, if any:

```
class Graph {
    int edges[][];
    public Graph(int[][] edges) { this.edges = edges; }

    public Graph loseEdges(int i, int j) {
        int n = edges.length;
        int[][] newedges = new int[n][];
        for (int k = 0; k < n; ++k) {

            edgelist: {
```



```

        int z;

        search: {
            if (k == i) {
                for (z = 0; z < edges[k].length; ++z)
                    if (edges[k][z] == j)
                        break search;
            } else if (k == j) {
                for (z = 0; z < edges[k].length; ++z)
                    if (edges[k][z] == i)
                        break search;
            }
            // No edge to be deleted; share this list.
            newedges[k] = edges[k];
            break edgelist;
        } //search
        // Copy the list, omitting the edge at position z.
        int m = edges[k].length - 1;
        int ne[] = new int[m];
        System.arraycopy(edges[k], 0, ne, 0, z);
        System.arraycopy(edges[k], z+1, ne, z, m-z);
        newedges[k] = ne;
    } //edgelist
}
return new Graph(newedges);
}
}

```

Note the use of two statement labels, `edgelist` and `search`, and the use of `break` statements. This allows the code that copies a list, omitting one edge, to be shared between two separate tests, the test for an edge from node *i* to node *j*, and the test for an edge from node *j* to node *i*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.14 The continue Statement

A `continue` statement may occur only in a `while`, `do`, or `for` statement; statements of these three kinds are called *iteration statements*. Control passes to the loop-continuation point of an iteration statement.

ContinueStatement:

```
continue Identifieropt ;
```

A `continue` statement with no label attempts to transfer control to the innermost enclosing `while`, `do`, or `for` statement; this statement, which is called the *continue target*, then immediately ends the current iteration and begins a new one. To be precise, such a `continue` statement always completes abruptly, the reason being a `continue` with no label. If no `while`, `do`, or `for` statement encloses the `continue` statement, a compile-time error occurs.

A `continue` statement with label *Identifier* attempts to transfer control to the enclosing labeled statement (§14.6) that has the same *Identifier* as its label; that statement, which is called the *continue target*, then immediately ends the current iteration and begins a new one. The `continue target` must be a `while`, `do`, or `for` statement or a compile-time error occurs. More precisely, a `continue` statement with label *Identifier* always completes abruptly, the reason being a `continue` with label *Identifier*. If no labeled statement with *Identifier* as its label contains the `continue` statement, a compile-time error occurs.

It can be seen, then, that a `continue` statement always completes abruptly.

See the descriptions of the `while` statement (§14.10), `do` statement (§14.11), and `for` statement (§14.12) for a discussion of the handling of abrupt termination because of `continue`.

The preceding descriptions say "attempts to transfer control" rather than just "transfers control" because if there are any `try` statements (§14.18) within the `continue target` whose `try` blocks contain the `continue` statement, then any `finally` clauses of those `try` statements are executed, in order, innermost to outermost, before control is transferred to the `continue target`. Abrupt completion of a `finally` clause can disrupt the transfer of control initiated by a `continue` statement.

In the `Graph` example in the preceding section, one of the `break` statements is used to finish execution of the entire body of the outermost `for` loop. This `break` can be replaced by a `continue` if the `for` loop itself is labeled:

```
class Graph {
    . . .
    public Graph loseEdges(int i, int j) {
        int n = edges.length;
        int[][] newedges = new int[n][];

        edgelists: for (int k = 0; k < n; ++k) {
            int z;

            search: {
                if (k == i) {
```



```

        . . .
    } else if (k == j) {
        . . .
    }
    newedges[k] = edges[k];
    continue edgelists;
} //search
    . . .
} //edgelists
return new Graph(newedges);
}
}

```

Which to use, if either, is largely a matter of programming style.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.15 The return Statement

A `return` statement returns control to the invoker of a method (§8.4, §15.11) or constructor (§8.6, §15.8).

ReturnStatement:

```
return Expressionopt ;
```

A `return` statement with no *Expression* must be contained in the body of a method that is declared, using the keyword `void`, not to return any value (§8.4), or in the body of a constructor (§8.6). A compile-time error occurs if a `return` statement appears within a static initializer (§8.5). A `return` statement with no *Expression* attempts to transfer control to the invoker of the method or constructor that contains it. To be precise, a `return` statement with no *Expression* always completes abruptly, the reason being a `return` with no value.

A `return` statement with an *Expression* must be contained in a method declaration that is declared to return a value (§8.4) or a compile-time error occurs. The *Expression* must denote a variable or value of some type *T*, or a compile-time error occurs. The type *T* must be assignable (§5.2) to the declared result type of the method, or a compile-time error occurs.

A `return` statement with an *Expression* attempts to transfer control to the invoker of the method that contains it; the value of the *Expression* becomes the value of the method invocation. More precisely, execution of such a `return` statement first evaluates the *Expression*. If the evaluation of the *Expression* completes abruptly for some reason, then the `return` statement completes abruptly for that reason. If evaluation of the *Expression* completes normally, producing a value *V*, then the `return` statement completes abruptly, the reason being a `return` with value *V*.

It can be seen, then, that a `return` statement always completes abruptly.

The preceding descriptions say "attempts to transfer control" rather than just "transfers control" because if there are any `try` statements (§14.18) within the method or constructor whose `try` blocks contain the `return` statement, then any `finally` clauses of those `try` statements will be executed, in order, innermost to outermost, before control is transferred to the invoker of the method or constructor. Abrupt completion of a `finally` clause can disrupt the transfer of control initiated by a `return` statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.16 The throw Statement

A `throw` statement causes an exception (§11) to be thrown. The result is an immediate transfer of control (§11.3) that may exit multiple statements and multiple constructor, static and field initializer evaluations, and method invocations until a `try` statement (§14.18) is found that catches the thrown value. If no such `try` statement is found, then execution of the thread (§17, §20.20) that executed the `throw` is terminated (§11.3) after invocation of the `UncaughtException` method (§20.21.31) for the thread group to which the thread belongs.

ThrowStatement:

```
throw Expression ;
```

The *Expression* in a `throw` statement must denote a variable or value of a reference type which is assignable (§5.2) to the type `Throwable`, or a compile-time error occurs. Moreover, at least one of the following three conditions must be true, or a compile-time error occurs:

- The exception is not a checked exception (§11.2)-specifically, one of the following situations is true:
 - The type of the *Expression* is the class `RuntimeException` or a subclass of `RuntimeException`.
 - The type of the *Expression* is the class `Error` or a subclass of `Error`.
- The `throw` statement is contained in the `try` block of a `try` statement (§14.18) and the type of the *Expression* is assignable (§5.2) to the type of the parameter of at least one `catch` clause of the `try` statement. (In this case we say the thrown value is *caught* by the `try` statement.)
- The `throw` statement is contained in a method or constructor declaration and the type of the *Expression* is assignable (§5.2) to at least one type listed in the `throws` clause (§8.4.4, §8.6.4) of the declaration.

A `throw` statement first evaluates the *Expression*. If the evaluation of the *Expression* completes abruptly for some reason, then the `throw` completes abruptly for that reason. If evaluation of the *Expression* completes normally, producing a value *V*, then the `throw` statement completes abruptly, the reason being a `throw` with value *V*.

It can be seen, then, that a `throw` statement always completes abruptly.

If there are any enclosing `try` statements (§14.18) whose `try` blocks contain the `throw` statement, then any `finally` clauses of those `try` statements are executed as control is transferred outward, until the thrown value is caught. Note that abrupt completion of a `finally` clause can disrupt the transfer of control initiated by a `throw` statement.

If a `throw` statement is contained in a method declaration, but its value is not caught by some `try` statement that contains it, then the invocation of the method completes abruptly because of the `throw`.

If a `throw` statement is contained in a constructor declaration, but its value is not caught by some `try` statement that contains it, then the class instance creation expression (or the method invocation of method `newInstance` of class `Class`) that invoked the constructor will complete abruptly because of the `throw`.

If a `throw` statement is contained in a static initializer (§8.5), then a compile-time check ensures that either its value is always an unchecked exception or its value is always caught by some `try` statement that contains it. If, despite this check, the value is not caught by some `try` statement that contains the `throw` statement, then the value is rethrown if it is an instance of class `Error` or one of its subclasses; otherwise, it is wrapped in an `ExceptionInInitializerError` object, which is then thrown (§12.4.2).

By convention, user-declared throwable types should usually be declared to be subclasses of class `Exception`, which is a subclass of class `Throwable` (§11.5, §20.22).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.17 The synchronized Statement

A `synchronized` statement acquires a mutual-exclusion lock (§17.13) on behalf of the executing thread, executes a block, then releases the lock. While the executing thread owns the lock, no other thread may acquire the lock.

SynchronizedStatement:

```
synchronized ( Expression ) Block
```

The type of *Expression* must be a reference type, or a compile-time error occurs.

A `synchronized` statement is executed by first evaluating the *Expression*.

If evaluation of the *Expression* completes abruptly for some reason, then the `synchronized` statement completes abruptly for the same reason.

Otherwise, if the value of the *Expression* is `null`, a `NullPointerException` is thrown.

Otherwise, let the non-`null` value of the *Expression* be *V*. The executing thread locks the lock associated with *V*. Then the *Block* is executed. If execution of the *Block* completes normally, then the lock is unlocked and the `synchronized` statement completes normally. If execution of the *Block* completes abruptly for any reason, then the lock is unlocked and the `synchronized` statement then completes abruptly for the same reason.

Acquiring the lock associated with an object does not of itself prevent other threads from accessing fields of the object or invoking unsynchronized methods on the object. Other threads can also use `synchronized` methods or the `synchronized` statement in a conventional manner to achieve mutual exclusion.

The locks acquired by `synchronized` statements are the same as the locks that are acquired implicitly by `synchronized` methods; see §8.4.3.5. A single thread may hold a lock more than once. The example:

```
class Test {
    public static void main(String[] args) {
        Test t = new Test();
        synchronized(t) {
            synchronized(t) {
                System.out.println("made it!");
            }
        }
    }
}
```

prints:

```
made it!
```

This example would deadlock if a single thread were not permitted to lock a lock more than once.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.18 The try statement

These are the times that try men's souls.

--Thomas Paine, *The American Crisis* (1780)

*. . . and they all fell to playing the game of catch as catch can,
till the gunpowder ran out at the heels of their boots.*

--Samuel Foote

A `try` statement executes a block. If a value is thrown and the `try` statement has one or more `catch` clauses that can catch it, then control will be transferred to the first such `catch` clause. If the `try` statement has a `finally` clause, then another block of code is executed, no matter whether the `try` block completes normally or abruptly, and no matter whether a `catch` clause is first given control.

TryStatement:

try Block Catches

try Block Catchesopt Finally

Catches:

CatchClause

Catches CatchClause

CatchClause:

catch (FormalParameter) Block

Finally:

finally Block

The following is repeated from §8.4.1 to make the presentation here clearer:

FormalParameter:

Type VariableDeclaratorId

The following is repeated from §8.3 to make the presentation here clearer:

VariableDeclaratorId:

Identifier

VariableDeclaratorId []

The *Block* immediately after the keyword `try` is called the `try` block of the `try` statement. The *Block* immediately after the keyword `finally` is called the `finally` block of the `try` statement.

A `try` statement may have `catch` clauses (also called *exception handlers*). A `catch` clause must have exactly one parameter (which is called an *exception parameter*); the declared type of the exception parameter must be the class `Throwable` or a subclass of `Throwable`, or a compile-time error occurs. The scope of the parameter variable is the *Block* of the `catch` clause. An exception parameter must not have the same name as a local variable or parameter in whose scope it is declared, or a compile-time error occurs.

The scope of the name of an exception parameter is the *Block* of the `catch` clause. The name of the parameter may not be redeclared as a local variable or exception parameter within the *Block* of the `catch` clause; that is, hiding the name of an exception parameter is not permitted.

Exception parameters cannot be referred to using qualified names ([§6.6](#)), only by simple names.

Exception handlers are considered in left-to-right order: the earliest possible `catch` clause accepts the exception, receiving as its actual argument the thrown exception object.

A `finally` clause ensures that the `finally` block is executed after the `try` block and any `catch` block that might be executed, no matter how control leaves the `try` block or `catch` block.

Handling of the `finally` block is rather complex, so the two cases of a `try` statement with and without a `finally` block are described separately.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.18.1 Execution of try-catch

Our supreme task is the resumption of our onward, normal way.

--Warren G. Harding, Inaugural Address (1921)

A `try` statement without a `finally` block is executed by first executing the `try` block. Then there is a choice:

- If execution of the `try` block completes normally, then no further action is taken and the `try` statement completes normally.
- If execution of the `try` block completes abruptly because of a `throw` of a value *V*, then there is a choice:
 - If the run-time type of *V* is assignable ([§5.2](#)) to the *Parameter* of any `catch` clause of the `try` statement, then the first (leftmost) such `catch` clause is selected. The value *V* is assigned to the parameter of the selected `catch` clause, and the *Block* of that `catch` clause is executed. If that block completes normally, then the `try` statement completes normally; if that block completes abruptly for any reason, then the `try` statement completes abruptly for the same reason.
 - If the run-time type of *V* is not assignable to the parameter of any `catch` clause of the `try` statement, then the `try` statement completes abruptly because of a `throw` of the value *V*.
- If execution of the `try` block completes abruptly for any other reason, then the `try` statement completes abruptly for the same reason.

In the example:

```
class BlewIt extends Exception {
    BlewIt() { }
    BlewIt(String s) { super(s); }
}
class Test {
    static void blowUp() throws BlewIt { throw new BlewIt(); }
    public static void main(String[] args) {

        try {
            blowUp();
        } catch (RuntimeException r) {
            System.out.println("RuntimeException:" + r);
        } catch (BlewIt b) {
            System.out.println("BlewIt");
        }

    }
}
```

the exception `BlewIt` is thrown by the method `blowUp`. The `try-catch` statement in the body of `main` has two `catch` clauses. The run-time type of the exception is `BlewIt` which is not assignable to a variable of type `RuntimeException`, but is assignable to a variable of type `BlewIt`, so the

output of the example is:

```
BlewIt
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.18.2 Execution of try-catch-finally

*After the great captains and engineers have accomplish'd their work,
After the noble inventors--after the scientists, the chemist,
the geologist, ethnologist,
Finally shall come the Poet . . .
--Walt Whitman, *Passage to India* (1870)*

A `try` statement with a `finally` block is executed by first executing the `try` block. Then there is a choice:

- If execution of the `try` block completes normally, then the `finally` block is executed, and then there is a choice:
 - If the `finally` block completes normally, then the `try` statement completes normally.
 - If the `finally` block completes abruptly for reason *S*, then the `try` statement completes abruptly for reason *S*.
- If execution of the `try` block completes abruptly because of a `throw` of a value *V*, then there is a choice:
 - If the run-time type of *V* is assignable to the parameter of any `catch` clause of the `try` statement, then the first (leftmost) such `catch` clause is selected. The value *V* is assigned to the parameter of the selected `catch` clause, and the *Block* of that `catch` clause is executed. Then there is a choice:
 - If the `catch` block completes normally, then the `finally` block is executed. Then there is a choice:
 - **If the `finally` block completes normally, then the `try` statement completes normally.**
 - **If the `finally` block completes abruptly for any reason, then the `try` statement completes abruptly for the same reason.**
 - If the `catch` block completes abruptly for reason *R*, then the `finally` block is executed. Then there is a choice:
 - **If the `finally` block completes normally, then the `try` statement completes abruptly for reason *R*.**
 - **If the `finally` block completes abruptly for reason *S*, then the `try` statement completes abruptly for reason *S* (and reason *R* is discarded).**
 - If the run-time type of *V* is not assignable to the parameter of any `catch` clause of the `try` statement, then the `finally` block is executed. Then there is a choice:
 - If the `finally` block completes normally, then the `try` statement completes abruptly because of a `throw` of the value *V*.
 - If the `finally` block completes abruptly for reason *S*, then the `try` statement completes abruptly for reason *S* (and the `throw` of value *V* is discarded and forgotten).
 - If execution of the `try` block completes abruptly for any other reason *R*, then the `finally` block is executed. Then there is a choice:
 - If the `finally` block completes normally, then the `try` statement completes abruptly for

reason *R*.

- If the `finally` block completes abruptly for reason *S*, then the `try` statement completes abruptly for reason *S* (and reason *R* is discarded).

The example:

```
class BlewIt extends Exception {
    BlewIt() { }
    BlewIt(String s) { super(s); }
}

class Test {

    static void blowUp() throws BlewIt {

        throw new NullPointerException();

    }

    public static void main(String[] args) {
        try {
            blowUp();
        } catch (BlewIt b) {
            System.out.println("BlewIt");
        } finally {
            System.out.println("Uncaught Exception");
        }
    }
}
```

produces the output:

```
Uncaught Exception
java.lang.NullPointerException
    at Test.blowUp(Test.java:7)
    at Test.main(Test.java:11)
```

The `NullPointerException` (which is a kind of `RuntimeException`) that is thrown by method `blowUp` is not caught by the `try` statement in `main`, because a `NullPointerException` is not assignable to a variable of type `BlewIt`. This causes the `finally` clause to execute, after which the thread executing `main`, which is the only thread of the test program, terminates because of an uncaught exception ([§20.21.31](#)), which results in printing the exception name and a simple `backtrace`.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


14.19 Unreachable Statements

That looks like a path.

Is that the way to reach the top from here?

--Robert Frost, *The Mountain* (1915)

It is a compile-time error if a statement cannot be executed because it is *unreachable*. Every Java compiler must carry out the conservative flow analysis specified here to make sure all statements are reachable.

This section is devoted to a precise explanation of the word "reachable." The idea is that there must be some possible execution path from the beginning of the constructor, method, or static initializer that contains the statement to the statement itself. The analysis takes into account the structure of statements. Except for the special treatment of `while`, `do`, and `for` statements whose condition expression has the constant value `true`, the values of expressions are not taken into account in the flow analysis. For example, a Java compiler will accept the code:

```
{
    int n = 5;
    while (n > 7) n = 2;
}
```

even though the value of `n` is known at compile time and in principle it can be known at compile time that the assignment to `n` can never be executed. A Java compiler must operate according to the rules laid out in this section.

The rules in this section define two technical terms:

- whether a statement is *reachable*
- whether a statement *can complete normally*

The definitions here allow a statement to complete normally only if it is reachable.

To shorten the description of the rules, the customary abbreviation "iff" is used to mean "if and only if."

The rules are as follows:

- The block that is the body of a constructor, method, or static initializer is reachable.
- An empty block that is not a switch block can complete normally iff it is reachable. A nonempty block that is not a switch block can complete normally iff the last statement in it can complete normally. The first statement in a nonempty block that is not a switch block is reachable iff the block is reachable. Every other statement `S` in a nonempty block that is not a switch block is reachable iff the statement preceding `S` can complete normally.
- A local variable declaration statement can complete normally iff it is reachable.
- An empty statement can complete normally iff it is reachable.
- A labeled statement can complete normally if at least one of the following is true:
 - The contained statement can complete normally.
 - There is a reachable `break` statement that exits the labeled statement.

The contained statement is reachable iff the labeled statement is reachable.

- An expression statement can complete normally iff it is reachable.
- The `if` statement, whether or not it has an `else` part, is handled in an unusual manner. For this reason, it is discussed separately at the end of this section.
- A `switch` statement can complete normally iff at least one of the following is true:
 - The last statement in the switch block can complete normally.
 - The switch block is empty or contains only switch labels.
 - There is at least one switch label after the last switch block statement group.
 - There is a reachable `break` statement that exits the `switch` statement.
- A switch block is reachable iff its `switch` statement is reachable.
- A statement in a switch block is reachable iff its `switch` statement is reachable and at least one of the following is true:
 - It bears a `case` or `default` label.
 - There is a statement preceding it in the `switch` block and that preceding statement can complete normally.
- A `while` statement can complete normally iff at least one of the following is true:
 - The `while` statement is reachable and the condition expression is not a constant expression with value `true`.
 - There is a reachable `break` statement that exits the `while` statement.

The contained statement is reachable iff the `while` statement is reachable and the condition expression is not a constant expression whose value is `false`.

- A `do` statement can complete normally iff at least one of the following is true:
 - The contained statement can complete normally and the condition expression is not a constant expression with value `true`.
 - There is a reachable `break` statement that exits the `do` statement.

The contained statement is reachable iff the `do` statement is reachable.

- A `for` statement can complete normally iff at least one of the following is true:
 - The `for` statement is reachable, there is a condition expression, and the condition expression is not a constant expression with value `true`.
 - There is a reachable `break` statement that exits the `for` statement.

The contained statement is reachable iff the `for` statement is reachable and the condition expression is not a constant expression whose value is `false`.

- A `break`, `continue`, `return`, or `throw` statement cannot complete normally.
- A `synchronized` statement can complete normally iff the contained statement can complete normally. The contained statement is reachable iff the `synchronized` statement is reachable.
- A `try` statement can complete normally iff both of the following are true:
 - The `try` block can complete normally or any `catch` block can complete normally.
 - If the `try` statement has a `finally` block, then the `finally` block can complete normally.
- The `try` block is reachable iff the `try` statement is reachable.
- A `catch` block `C` is reachable iff both of the following are true:
 - Some expression or `throw` statement in the `try` block is reachable and can throw an

exception whose type is assignable to the parameter of the `catch` clause *C*. (An expression is considered reachable iff the innermost statement containing it is reachable.)

- There is no earlier `catch` block *A* in the `try` statement such that the type of *C*'s parameter is the same as or a subclass of the type of *A*'s parameter.
- If a `finally` block is present, it is reachable iff the `try` statement is reachable.

One might expect the `if` statement to be handled in the following manner, but these are not the rules that Java actually uses:

- HYPOTHETICAL: An `if-then` statement can complete normally iff at least one of the following is `true`:
 - The `if-then` statement is reachable and the condition expression is not a constant expression whose value is `true`.
 - The `then-statement` can complete normally.

The `then-statement` is reachable iff the `if-then` statement is reachable and the condition expression is not a constant expression whose value is `false`.

- HYPOTHETICAL: An `if-then-else` statement can complete normally iff the `then-statement` can complete normally or the `else-statement` can complete normally. The `then-statement` is reachable iff the `if-then-else` statement is reachable and the condition expression is not a constant expression whose value is `false`. The `else` statement is reachable iff the `if-then-else` statement is reachable and the condition expression is not a constant expression whose value is `true`.

This approach would be consistent with the treatment of other control structures in Java. However, in order to allow the `if` statement to be used conveniently for "conditional compilation" purposes, the actual rules are as follows:

- ACTUAL: An `if-then` statement can complete normally iff it is reachable. The `then-statement` is reachable iff the `if-then` statement is reachable.
- ACTUAL: An `if-then-else` statement can complete normally iff the `then-statement` can complete normally or the `else-statement` can complete normally. The `then-statement` is reachable iff the `if-then-else` statement is reachable. The `else-statement` is reachable iff the `if-then-else` statement is reachable.

As an example, the following statement results in a compile-time error:

```
while (false) { x=3; }
```

because the statement `x=3;` is not reachable; but the superficially similar case:

```
if (false) { x=3; }
```

does not result in a compile-time error. An optimizing compiler may realize that the statement `x=3;` will never be executed and may choose to omit the code for that statement from the generated `class` file, but the statement `x=3;` is not regarded as "unreachable" in the technical sense specified here.

The rationale for this differing treatment is to allow programmers to define "flag variables" such as:


```
static final boolean DEBUG = false;
```

and then write code such as:

```
if (DEBUG) { x=3; }
```

The idea is that it should be possible to change the value of `DEBUG` from `false` to `true` or from `true` to `false` and then compile the code correctly with no other changes to the program text.

This ability to "conditionally compile" has a significant impact on, and relationship to, binary compatibility (§13). If a set of classes that use such a "flag" variable are compiled and conditional code is omitted, it does not suffice later to distribute just a new version of the class or interface that contains the definition of the flag. A change to the value of a flag is, therefore, not binary compatible with preexisting binaries (§13.4.8). (There are other reasons for such incompatibility as well, such as the use of constants in `case` labels in `switch` statements; see §13.4.8.)

One ought not to be thrown into confusion

By a plain statement of relationship . . .

—Robert Frost, *The Generations of Men* (1914)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Expressions

*When you can measure what you are speaking about,
and express it in numbers, you know something about it;
but when you cannot measure it, when you cannot express it in numbers,
your knowledge of it is of a meager and unsatisfactory kind:
it may be the beginning of knowledge, but you have scarcely,
in your thoughts, advanced to the stage of science.*

--William Thompson, Lord Kelvin

Much of the work in a Java program is done by evaluating *expressions*, either for their side effects, such as assignments to variables, or for their values, which can be used as arguments or operands in larger expressions, or to affect the execution sequence in statements, or both.

This chapter specifies the meanings of Java expressions and the rules for their evaluation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.1 Evaluation, Denotation, and Result

When an expression in a Java program is *evaluated* (*executed*), the *result* denotes one of three things:

- A variable (§4.5) (in C, this would be called an *lvalue*)
- A value (§4.2, §4.3)
- Nothing (the expression is said to be `void`)

Evaluation of an expression can also produce side effects, because expressions may contain embedded assignments, increment operators, decrement operators, and method invocations.

An expression denotes nothing if and only if it is a method invocation (§15.11) that invokes a method that does not return a value, that is, a method declared `void` (§8.4). Such an expression can be used only as an expression statement (§14.7), because every other context in which an expression can appear requires the expression to denote something. An expression statement that is a method invocation may also invoke a method that produces a result; in this case the value returned by the method is quietly discarded.

Each expression occurs in the declaration of some (class or interface) type that is being declared: in a field initializer, in a static initializer, in a constructor declaration, or in the code for a method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.2 Variables as Values

If an expression denotes a variable, and a value is required for use in further evaluation, then the value of that variable is used. In this context, if the expression denotes a variable or a value, we may speak simply of the *value* of the expression.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.3 Type of an Expression

If an expression denotes a variable or a value, then the expression has a type known at compile time. The rules for determining the type of an expression are explained separately below for each kind of expression.

The value of an expression is always assignment compatible ([§5.2](#)) with the type of the expression, just as the value stored in a variable is always compatible with the type of the variable. In other words, the value of an expression whose type is T is always suitable for assignment to a variable of type T .

Note that an expression whose type is a class type F that is declared `final` is guaranteed to have a value that is either a null reference or an object whose class is F itself, because `final` types have no subclasses.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.4 Expressions and Run-Time Checks

If the type of an expression is a primitive type, then the value of the expression is of that same primitive type. But if the type of an expression is a reference type, then the class of the referenced object, or even whether the value is a reference to an object rather than `null`, is not necessarily known at compile time. There are a few places in the Java language where the actual class of a referenced object affects program execution in a manner that cannot be deduced from the type of the expression. They are as follows:

- Method invocation ([§15.11](#)). The particular method used for an invocation `o.m(...)` is chosen based on the methods that are part of the class or interface that is the type of `o`. For instance methods, the class of the object referenced by the run-time value of `o` participates because a subclass may override a specific method already declared in a parent class so that this overriding method is invoked. (The overriding method may or may not choose to further invoke the original overridden `m` method.)
- The `instanceof` operator ([§15.19.2](#)). An expression whose type is a reference type may be tested using `instanceof` to find out whether the class of the object referenced by the run-time value of the expression is assignment compatible ([§5.2](#)) with some other reference type.
- Casting ([§5.4](#), [§15.15](#)). The class of the object referenced by the run-time value of the operand expression might not be compatible with the type specified by the cast. For reference types, this may require a run-time check that throws an error if the class of the referenced object, as determined at run time, is not assignment compatible ([§5.2](#)) with the target type.
- Assignment to an array component of reference type ([§10.10](#), [§15.12](#), [§15.25.1](#)). The type-checking rules allow the array type `S[]` to be treated as a subtype of `T[]` if `S` is a subtype of `T`, but this requires a run-time check for assignment to an array component, similar to the check performed for a cast.
- Exception handling ([§14.18](#)). An exception is caught by a `catch` clause only if the class of the thrown exception object is an `instanceof` the type of the formal parameter of the `catch` clause.

The first two of the cases just listed ought never to result in detecting a type error. Thus, a Java run-time type error can occur only in these situations:

- In a cast, when the actual class of the object referenced by the value of the operand expression is not compatible with the target type specified by the cast operator ([§5.4](#), [§15.15](#)); in this case a `ClassCastException` is thrown.
- In an assignment to an array component of reference type, when the actual class of the object referenced by the value to be assigned is not compatible with the actual run-time component type of the array ([§10.10](#), [§15.12](#), [§15.25.1](#)); in this case an `ArrayStoreException` is thrown.
- When an exception is not caught by any `catch` handler ([§11.3](#)); in this case the thread of control that encountered the exception first invokes the method `uncaughtException` ([§20.21.31](#)) for its thread group and then terminates.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.5 Normal and Abrupt Completion of Evaluation

No more: the end is sudden and abrupt.

--William Wordsworth, Apology for the Foregoing Poems (1831)

Every expression has a normal mode of evaluation in which certain computational steps are carried out. The following sections describe the normal mode of evaluation for each kind of expression. If all the steps are carried out without an exception being thrown, the expression is said to *complete normally*.

If, however, evaluation of an expression throws an exception, then the expression is said to *complete abruptly*. An abrupt completion always has an associated *reason*, which is always a `throw` with a given value.

Run-time exceptions are thrown by the predefined operators as follows:

- A class instance creation expression (§15.8), array creation expression (§15.9), or string concatenation operator expression (§15.17.1) throws an `OutOfMemoryError` if there is insufficient memory available.
- An array creation expression throws an `ArrayNegativeSizeException` if the value of any dimension expression is less than zero (§15.9).
- A field access (§15.10) throws a `NullPointerException` if the value of the object reference expression is `null`.
- A method invocation expression (§15.11) that invokes an instance method throws a `NullPointerException` if the target reference is `null`.
- An array access (§15.12) throws a `NullPointerException` if the value of the array reference expression is `null`.
- An array access (§15.12) throws an `IndexOutOfBoundsException` if the value of the array index expression is negative or greater than or equal to the `length` of the array.
- A cast (§15.15) throws a `ClassCastException` if a cast is found to be impermissible at run time.
- An integer division (§15.16.2) or integer remainder (§15.16.3) operator throws an `ArithmeticException` if the value of the right-hand operand expression is zero.
- An assignment to an array component of reference type (§15.25.1) throws an `ArrayStoreException` when the value to be assigned is not compatible with the component type of the array.

A method invocation expression can also result in an exception being thrown if an exception occurs that causes execution of the method body to complete abruptly. A class instance creation expression can also result in an exception being thrown if an exception occurs that causes execution of the constructor to complete abruptly. Various linkage and virtual machine errors may also occur during the evaluation of an expression. By their nature, such errors are difficult to predict and difficult to handle.

If an exception occurs, then evaluation of one or more expressions may be terminated before all steps of their normal mode of evaluation are complete; such expressions are said to complete abruptly. The terms "complete normally" and "complete abruptly" are also applied to the execution of statements (§14.1). A statement may complete abruptly for a variety of reasons, not just because an exception is thrown.

If evaluation of an expression requires evaluation of a subexpression, abrupt completion of the

subexpression always causes the immediate abrupt completion of the expression itself, with the same reason, and all succeeding steps in the normal mode of evaluation are not performed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.6 Evaluation Order

Let all things be done decently and in order.

--I Corinthians 14:40

Java guarantees that the operands of operators appear to be evaluated in a specific *evaluation order*, namely, from left to right.

It is recommended that Java code not rely crucially on this specification. Code is usually clearer when each expression contains at most one side effect, as its outermost operation, and when code does not depend on exactly which exception arises as a consequence of the left-to-right evaluation of expressions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.6.1 Left-Hand Operand First

The left-hand operand of a binary operator appears to be fully evaluated before any part of the right-hand operand is evaluated. For example, if the left-hand operand contains an assignment to a variable and the right-hand operand contains a reference to that same variable, then the value produced by the reference will reflect the fact that the assignment occurred first.

Thus:

```
class Test {
    public static void main(String[] args) {
        int i = 2;
        int j = (i=3) * i;
        System.out.println(j);
    }
}
```

prints:

9

It is not permitted for it to print 6 instead of 9.

If the operator is a compound-assignment operator ([§15.25.2](#)), then evaluation of the left-hand operand includes both remembering the variable that the left-hand operand denotes and fetching and saving that variable's value for use in the implied combining operation. So, for example, the test program:

```
class Test {
    public static void main(String[] args) {
        int a = 9;
        a += (a = 3);
        // first example
        System.out.println(a);
        int b = 9;
        b = b + (b = 3);
        // second example
        System.out.println(b);
    }
}
```

prints:

12
12

because the two assignment statements both fetch and remember the value of the left-hand

operand, which is 9, before the right-hand operand of the addition is evaluated, thereby setting the variable to 3. It is not permitted for either example to produce the result 6. Note that both of these examples have unspecified behavior in C, according to the ANSI/ISO standard.

If evaluation of the left-hand operand of a binary operator completes abruptly, no part of the right-hand operand appears to have been evaluated.

Thus, the test program:

```
class Test {  
  
    public static void main(String[] args) {  
  
        int j = 1;  
  
        try {  
            int i = forgetIt() / (j = 2);  
        } catch (Exception e) {  
            System.out.println(e);  
            System.out.println("Now j = " + j);  
        }  
  
        static int forgetIt() throws Exception {  
            throw new Exception("I'm outta here!");  
        }  
    }  
}
```

prints:

```
java.lang.Exception: I'm outta here!  
Now j = 1
```

because the left-hand operand `forgetIt()` of the operator `/` throws an exception before the right-hand operand and its embedded assignment of 2 to `j` occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.6.2 Operands before Operation

Java also guarantees that every operand of an operator (except the conditional operators `&&`, `||`, and `? :`) appears to be fully evaluated before any part of the operation itself is performed.

If the binary operator is an integer division `/` ([§15.16.2](#)) or integer remainder `%` ([§15.16.3](#)), then its execution may raise an `ArithmeticException`, but this exception is thrown only after both operands of the binary operator have been evaluated and only if these evaluations completed normally.

So, for example, the program:

```
class Test {  
  
    public static void main(String[] args) {  
        int divisor = 0;  
        try {  
            int i = 1 / (divisor * loseBig());  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
  
    static int loseBig() throws Exception {  
        throw new Exception("Shuffle off to Buffalo!");  
    }  
}
```

always prints:

```
java.lang.Exception: Shuffle off to Buffalo!
```

and not:

```
java.lang.ArithmeticException: / by zero
```

since no part of the division operation, including signaling of a divide-by-zero exception, may appear to occur before the invocation of `loseBig` completes, even though the implementation may be able to detect or infer that the division operation would certainly result in a divide-by-zero exception.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.6.3 Parentheses and Precedence Respected

That is too weighty a subject to be discussed parenthetically . . .

--John Stuart Mill, *On Liberty* (1869), Chapter IV

Java implementations must respect the order of evaluation as indicated explicitly by parentheses and implicitly by operator precedence. An implementation may not take advantage of algebraic identities such as the associative law to rewrite expressions into a more convenient computational order unless it can be proven that the replacement expression is equivalent in value and in its observable side effects, even in the presence of multiple threads of execution (using the thread execution model in [§17](#)), for all possible computational values that might be involved.

In the case of floating-point calculations, this rule applies also for infinity and not-a-number (NaN) values. For example, `!(x < y)` may not be rewritten as `x >= y`, because these expressions have different values if either `x` or `y` is NaN.

Specifically, floating-point calculations that appear to be mathematically associative are unlikely to be computationally associative. Such computations must not be naively reordered. For example, it is not correct for a Java compiler to rewrite `4.0 * x * 0.5` as `2.0 * x`; while roundoff happens not to be an issue here, there are large values of `x` for which the first expression produces infinity (because of overflow) but the second expression produces a finite result.

So, for example, the test program:

```
class Test {  
    public static void main(String[] args) {  
        double d = 8e+307;  
        System.out.println(4.0 * d * 0.5);  
        System.out.println(2.0 * d);  
    }  
}
```

prints:

```
Infinity  
1.6e+308
```

because the first expression overflows and the second does not.

In contrast, integer addition and multiplication *are* provably associative in Java; for example `a+b+c`, where `a`, `b`, and `c` are local variables (this simplifying assumption avoids issues involving multiple threads and `volatile` variables), will always produce the same answer whether evaluated as `(a+b)+c` or `a+(b+c)`; if the expression `b+c` occurs nearby in the code, a smart compiler may be able to use this common subexpression.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.6.4 Argument Lists Evaluated Left-to-Right

In a method or constructor invocation or class instance creation expression, argument expressions may appear within the parentheses, separated by commas. Each argument expression appears to be fully evaluated before any part of any argument expression to its right.

Thus:

```
class Test {

    public static void main(String[] args) {
        String s = "going, ";
        print3(s, s, s = "gone");
    }

    static void print3(String a, String b, String c) {
        System.out.println(a + b + c);
    }
}
```

always prints:

going, going, gone

because the assignment of the string "gone" to `s` occurs after the first two arguments to `print3` have been evaluated.

If evaluation of an argument expression completes abruptly, no part of any argument expression to its right appears to have been evaluated.

Thus, the example:

```
class Test {
    static int id;
    public static void main(String[] args) {
        try {
            test(id = 1, oops(), id = 3);
        } catch (Exception e) {
            System.out.println(e + ", id=" + id);
        }
    }

    static int oops() throws Exception {
        throw new Exception("oops");
    }

    static int test(int a, int b, int c) {
        return a + b + c;
    }
}
```



```
    }  
}
```

prints:

```
java.lang.Exception: oops, id=1
```

because the assignment of 3 to `id` is not executed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.6.5 Ordering of Other Expressions

The order of evaluation for some expressions is not completely covered by these general rules, because these expressions may raise exceptional conditions at times that must be specified. See, specifically, the detailed explanations of evaluation order for the following kinds of expressions:

- class instance creation expressions ([§15.8.1](#))
- array creation expressions ([§15.9.1](#))
- method invocation expressions ([§15.11.4](#))
- array access expressions ([§15.12.1](#))
- assignments involving array components ([§15.25](#))

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.7 Primary Expressions

Primary expressions include most of the simplest kinds of expressions, from which all others are constructed: literals, field accesses, method invocations, and array accesses. A parenthesized expression is also treated syntactically as a primary expression.

Primary:

PrimaryNoNewArray

ArrayCreationExpression

PrimaryNoNewArray:

Literal

this

(Expression)

ClassInstanceCreationExpression

FieldAccess

MethodInvocation

ArrayAccess

As programming language grammars go, this part of the Java grammar is unusual, in two ways. First, one might expect simple names, such as names of local variables and method parameters, to be primary expressions. For technical reasons, names are lumped together with primary expressions a little later when postfix expressions are introduced ([§15.13](#)).

The technical reasons have to do with allowing left-to-right parsing of Java programs with only one-token lookahead. Consider the expressions `(z [3])` and `(z [])`. The first is a parenthesized array access ([§15.12](#)) and the second is the start of a cast ([§15.15](#)). At the point that the look-ahead symbol is `[`, a left-to-right parse will have reduced the `z` to the nonterminal *Name*. In the context of a cast we prefer not to have to reduce the name to a *Primary*, but if *Name* were one of the alternatives for *Primary*, then we could not tell whether to do the reduction (that is, we could not determine whether the current situation would turn out to be a parenthesized array access or a cast) without looking ahead two tokens, to the token following the `[`. The Java grammar presented here avoids the problem by keeping *Name* and *Primary* separate and allowing either in certain other syntax rules (those for *MethodInvocation*, *ArrayAccess*, *PostfixExpression*, but not for *FieldAccess*, because this is covered by *Name*). This strategy effectively defers the question of whether a *Name* should be treated as a *Primary* until more context can be examined. (Other problems remain with cast expressions; see [§19.1.5](#).)

The second unusual feature avoids a potential grammatical ambiguity in the expression:

```
new int[3][3]
```


which in Java always means a single creation of a multidimensional array, but which, without appropriate grammatical finesse, might also be interpreted as meaning the same as:

```
(new int[3])[3]
```

This ambiguity is eliminated by splitting the expected definition of *Primary* into *Primary* and *PrimaryNoNewArray*. (This may be compared to the splitting of *Statement* into *Statement* and *StatementNoShortIf* (§14.4) to avoid the "dangling `else`" problem.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.7.1 Literals

A literal (§3.10) denotes a fixed, unchanging value.

The following production from §3.10 is repeated here for convenience:

Literal:

IntegerLiteral

FloatingPointLiteral

BooleanLiteral

CharacterLiteral

StringLiteral

NullLiteral

The type of a literal is determined as follows:

- The type of an integer literal that ends with `L` or `l` is `long`; the type of any other integer literal is `int`.
- The type of a floating-point literal that ends with `F` or `f` is `float`; the type of any other floating-point literal is `double`.
- The type of a boolean literal is `boolean`.
- The type of a character literal is `char`.
- The type of a string literal is `String`.
- The type of the null literal `null` is the null type; its value is the null reference.

Evaluation of a literal always completes normally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.7.2 this

The keyword `this` may be used only in the body of an instance method or constructor, or in the initializer of an instance variable of a class. If it appears anywhere else, a compile-time error occurs.

When used as a primary expression, the keyword `this` denotes a value, that is a reference to the object for which the instance method was invoked ([§15.11](#)), or to the object being constructed. The type of `this` is the class `C` within which the keyword `this` occurs. At run time, the class of the actual object referred to may be the class `C` or any subclass of `C`.

In the example:

```
class IntVector {
    int[] v;
    boolean equals(IntVector other) {
        if (this == other)
            return true;
        if (v.length != other.v.length)
            return false;
        for (int i = 0; i < v.length; i++)
            if (v[i] != other.v[i])
                return false;
        return true;
    }
}
```

the class `IntVector` implements a method `equals`, which compares two vectors. If the `other` vector is the same vector object as the one for which the `equals` method was invoked, then the check can skip the length and value comparisons. The `equals` method implements this check by comparing the reference to the `other` object to `this`.

The keyword `this` is also used in a special explicit constructor invocation statement, which can appear at the beginning of a constructor body ([§8.6.5](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.7.3 Parenthesized Expressions

A parenthesized expression is a primary expression whose type is the type of the contained expression and whose value at run time is the value of the contained expression.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.8 Class Instance Creation Expressions

A class instance creation expression is used to create new objects that are instances of classes.

ClassInstanceCreationExpression:

new ClassType (ArgumentListopt)

ArgumentList:

Expression

ArgumentList , Expression

In a class instance creation expression, the *ClassType* must name a class that is not `abstract`. This class type is the type of the creation expression.

The arguments in the argument list, if any, are used to select a constructor declared in the body of the named class type, using the same matching rules as for method invocations ([§15.11](#)). As in method invocations, a compile-time method matching error results if there is no unique constructor that is both applicable to the provided arguments and the most specific of all the applicable constructors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.8.1 Run-time Evaluation of Class Instance Creation Expressions

At run time, evaluation of a class instance creation expression is as follows.

First, space is allocated for the new class instance. If there is insufficient space to allocate the object, evaluation of the class instance creation expression completes abruptly by throwing an

`OutOfMemoryError` (§15.8.2).

The new object contains new instances of all the fields declared in the specified class type and all its superclasses. As each new field instance is created, it is initialized to its standard default value (§4.5.4).

Next, the argument list is evaluated, left-to-right. If any of the argument evaluations completes abruptly, any argument expressions to its right are not evaluated, and the class instance creation expression completes abruptly for the same reason.

Next, the selected constructor of the specified class type is invoked. This results in invoking at least one constructor for each superclass of the class type. This process can be directed by explicit constructor invocation statements (§8.6) and is described in detail in §12.5.

The value of a class instance creation expression is a reference to the newly created object of the specified class. Every time the expression is evaluated, a fresh object is created.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.8.2 Example: Evaluation Order and Out-of-Memory Detection

If evaluation of a class instance creation expression finds there is insufficient memory to perform the creation operation, then an `OutOfMemoryError` is thrown. This check occurs before any argument expressions are evaluated.

So, for example, the test program:

```
class List {
    int value;
    List next;
    static List head = new List(0);
    List(int n) { value = n; next = head; head = this; }
}

class Test {
    public static void main(String[] args) {
        int id = 0, oldid = 0;
        try {
            for (;;) {
                ++id;
                new List(oldid = id);
            }
        } catch (Error e) {
            System.out.println(e + ", " + (oldid==id));
        }
    }
}
```

prints:

```
java.lang.OutOfMemoryError: List, false
```

because the out-of-memory condition is detected before the argument expression `oldid = id` is evaluated.

Compare this to the treatment of array creation expressions ([§15.9](#)), for which the out-of-memory condition is detected after evaluation of the dimension expressions ([§15.9.3](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.9 Array Creation Expressions

An array instance creation expression is used to create new arrays (§10).

ArrayCreationExpression:

new PrimitiveType DimExprs Dimsopt

new TypeName DimExprs Dimsopt

DimExprs:

DimExpr

DimExprs DimExpr

DimExpr:

[Expression]

Dims:

[]

Dims []

An array creation expression creates an object that is a new array whose elements are of the type specified by the *PrimitiveType* or *TypeName*. The *TypeName* may name any reference type, even an abstract class type (§8.1.2.1) or an interface type (§9).

The type of the creation expression is an array type that can denoted by a copy of the creation expression from which the `new` keyword and every *DimExpr* expression have been deleted; for example, the type of the creation expression:

```
new double[3][3][ ]
```

is:

```
double[ ][ ][ ]
```

The type of each dimension expression *DimExpr* must be an integral type, or a compile-time error occurs. Each expression undergoes unary numeric promotion (§5.6.1). The promoted type must be `int`, or a compile-time error occurs; this means, specifically, that the type of a dimension expression must not be `long`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.9.1 Run-time Evaluation of Array Creation Expressions

At run time, evaluation of an array creation expression behaves as follows.

First, the dimension expressions are evaluated, left-to-right. If any of the expression evaluations completes abruptly, the expressions to the right of it are not evaluated.

Next, the values of the dimension expressions are checked. If the value of any *DimExpr* expression is less than zero, then an `NegativeArraySizeException` is thrown.

Next, space is allocated for the new array. If there is insufficient space to allocate the array, evaluation of the array creation expression completes abruptly by throwing an `OutOfMemoryError`.

Then, if a single *DimExpr* appears, a single-dimensional array is created of the specified length, and each component of the array is initialized to its standard default value (§4.5.4).

If an array creation expression contains *N DimExpr* expressions, then it effectively executes a set of nested loops of depth *N* to create the implied arrays of arrays. For example, the declaration:

```
float[][] matrix = new float[3][3];
```

is equivalent in behavior to:

```
float[][] matrix = new float[3][];
for (int d = 0; d < matrix.length; d++)
    matrix[d] = new float[3];
```

and:

```
Age[][][][][] Aquarius = new Age[6][10][8][12][];
```

is equivalent to:

```
Age[][][][][] Aquarius = new Age[6][][][][];
for (int d1 = 0; d1 < Aquarius.length; d1++) {
    Aquarius[d1] = new Age[8][][][];
    for (int d2 = 0; d2 < Aquarius[d1].length; d2++) {
        Aquarius[d1][d2] = new Age[10][][];
        for (int d3 = 0; d3 < Aquarius[d1][d2].length; d3++) {
            Aquarius[d1][d2][d3] = new Age[12][];
        }
    }
}
```

with *d*, *d1*, *d2* and *d3* replaced by names that are not already locally declared. Thus, a single `new` expression actually creates one array of length 6, 6 arrays of length 10, {ewc msdncl, EWGraphic, LNG20s 1 /a "langref5ANC1.BMP"} arrays of length 8, and {ewc msdncl,

EWGraphic, LNG20s 2 /a "langref5ANC2.BMP"} arrays of length 12. This example leaves the fifth dimension, which would be arrays containing the actual array elements (references to `Age` objects), initialized only to null references. These arrays can be filled in later by other code, such as:

```
Age[] Hair = { new Age("quartz"), new Age("topaz") };
Aquarius[1][9][6][9] = Hair;
```

A multidimensional array need not have arrays of the same length at each level; thus, a triangular matrix may be created by:

```
float triang[][] = new float[100][];
for (int i = 0; i < triang.length; i++)
    triang[i] = new float[i+1];
```

There is, however, no way to get this effect with a single creation expression.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.9.2 Example: Evaluation Order

In an array creation expression ([§15.9](#)), there may be one or more dimension expressions, each within brackets. Each dimension expression is fully evaluated before any part of any dimension expression to its right.

Thus:

```
class Test {
    public static void main(String[] args) {
        int i = 4;
        int ia[][] = new int[i][i=3];
        System.out.println(
            "[" + ia.length + "," + ia[0].length + "]"");
    }
}
```

prints:

[4,3]

because the first dimension is calculated as 4 before the second dimension expression sets `i` to 3.

If evaluation of a dimension expression completes abruptly, no part of any dimension expression to its right will appear to have been evaluated. Thus, the example:

```
class Test {

    public static void main(String[] args) {
        int[][] a = { { 00, 01 }, { 10, 11 } };
        int i = 99;
        try {
            a[val()][i = 1]++;
        } catch (Exception e) {
            System.out.println(e + ", i=" + i);
        }
    }

    static int val() throws Exception {
        throw new Exception("unimplemented");
    }
}
```

prints:

java.lang.Exception: unimplemented, i=99

because the embedded assignment that sets `i` to `1` is never executed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.9.3 Example: Evaluation Order and Out-of-Memory Detection

If evaluation of an array creation expression finds there is insufficient memory to perform the creation operation, then an `OutOfMemoryError` is thrown. This check occurs only after evaluation of all dimension expressions has completed normally.

So, for example, the test program:

```
class Test {
    public static void main(String[] args) {
        int len = 0, oldlen = 0;
        Object[] a = new Object[0];
        try {
            for (;;) {
                ++len;
                Object[] temp = new Object[oldlen = len];
                temp[0] = a;
                a = temp;
            }
        } catch (Error e) {
            System.out.println(e + ", " + (oldlen==len));
        }
    }
}
```

prints:

```
java.lang.OutOfMemoryError, true
```

because the out-of-memory condition is detected after the argument expression `oldlen = len` is evaluated.

Compare this to class instance creation expressions ([§15.8](#)), which detect the out-of-memory condition before evaluating argument expressions ([§15.8.2](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.10 Field Access Expressions

A field access expression may access a field of an object or array, a reference to which is the value of either an expression or the special keyword `super`. (It is also possible to refer to a field of the current instance or current class by using a simple name; see [§15.13.1](#).)

FieldAccess:

Primary . Identifier

super . Identifier

The meaning of a field access expression is determined using the same rules as for qualified names ([§6.6](#)), but limited by the fact that an expression cannot denote a package, class type, or interface type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.10.1 Field Access Using a Primary

The type of the *Primary* must be a reference type *T*, or a compile-time error occurs. The meaning of the field access expression is determined as follows:

- If the identifier names several accessible member fields of type *T*, then the field access is ambiguous and a compile-time error occurs.
- If the identifier does not name an accessible member field of type *T*, then the field access is undefined and a compile-time error occurs.
- Otherwise, the identifier names a single accessible member field of type *T* and the type of the field access expression is the declared type of the field. At run time, the result of the field access expression is computed as follows:
 - If the field is `static`:
 - If the field is `final`, then the result is the value of the specified class variable in the class or interface that is the type of the *Primary* expression.
 - If the field is not `final`, then the result is a variable, namely, the specified class variable in the class that is the type of the *Primary* expression.
 - If the field is not `static`:
 - If the value of the *Primary* is `null`, then a `NullPointerException` is thrown.
 - If the field is `final`, then the result is the value of the specified instance variable in the object referenced by the value of the *Primary*.
 - If the field is not `final`, then the result is a variable, namely, the specified instance variable in the object referenced by the value of the *Primary*.

Note, specifically, that only the type of the *Primary* expression, not the class of the actual object referred to at run time, is used in determining which field to use.

Thus, the example:

```
class S { int x = 0; }
class T extends S { int x = 1; }
class Test {
    public static void main(String[] args) {

        T t = new T();
        System.out.println("t.x=" + t.x + when("t", t));

        S s = new S();
        System.out.println("s.x=" + s.x + when("s", s));

        s = t;
        System.out.println("s.x=" + s.x + when("s", s));

    }
```



```

        static String when(String name, Object t) {
            return " when " + name + " holds a "
                + t.getClass() + " at run time.";
        }
    }
}

```

produces the output:

```

t.x=1 when t holds a class T at run time.
s.x=0 when s holds a class S at run time.
s.x=0 when s holds a class T at run time.

```

The last line shows that, indeed, the field that is accessed does not depend on the run-time class of the referenced object; even if `s` holds a reference to an object of class `T`, the expression `s.x` refers to the `x` field of class `S`, because the type of the expression `s` is `S`. Objects of class `T` contain two fields named `x`, one for class `T` and one for its superclass `S`.

This lack of dynamic lookup for field accesses allows Java to run efficiently with straightforward implementations. The power of late binding and overriding is available in Java, but only when instance methods are used. Consider the same example using instance methods to access the fields:

```

class S { int x = 0; int z() { return x; } }
class T extends S { int x = 1; int z() { return x; } }
class Test {

    public static void main(String[] args) {
        T t = new T();
        System.out.println("t.z()=" + t.z() + when("t", t));
        S s = new S();
        System.out.println("s.z()=" + s.z() + when("s", s));
        s = t;
        System.out.println("s.z()=" + s.z() + when("s", s));
    }

    static String when(String name, Object t) {
        return " when " + name + " holds a "
            + t.getClass() + " at run time.";
    }
}

```

Now the output is:

```

t.z()=1 when t holds a class T at run time.
s.z()=0 when s holds a class S at run time.
s.z()=1 when s holds a class T at run time.

```


The last line shows that, indeed, the method that is accessed *does* depend on the run-time class of referenced object; when `s` holds a reference to an object of class `T`, the expression `s.z()` refers to the `z` method of class `T`, despite the fact that the type of the expression `s` is `S`. Method `z` of class `T` overrides method `z` of class `S`.

The following example demonstrates that a null reference may be used to access a class (static) variable without causing an exception:

```
class Test {
    static String mountain = "Chocorua";

    static Test favorite(){
        System.out.print("Mount ");
        return null;
    }

    public static void main(String[] args) {
        System.out.println(favorite().mountain);
    }
}
```

It compiles, executes, and prints:

```
Mount Chocorua
```

Even though the result of `favorite()` is null, a `NullPointerException` is *not* thrown. That "Mount " is printed demonstrates that the *Primary* expression is indeed fully evaluated at run time, despite the fact that only its type, not its value, is used to determine which field to access (because the field `mountain` is static).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.10.2 Accessing Superclass Members using super

The special form using the keyword `super` is valid only in an instance method or constructor, or in the initializer of an instance variable of a class; these are exactly the same situations in which the keyword `this` may be used (§15.7.2). The form involving `super` may not be used anywhere in the class `Object`, since `Object` has no superclass; if `super` appears in class `Object`, then a compile-time error results.

Suppose that a field access expression `super.name` appears within class `C`, and the immediate superclass of `C` is class `S`. Then `super.name` is treated exactly as if it had been the expression `((S) this).name`; thus, it refers to the field named `name` of the current object, but with the current object viewed as an instance of the superclass. Thus it can access the field named `name` that is visible in class `S`, even if that field is hidden by a declaration of a field named `name` in class `C`.

The use of `super` is demonstrated by the following example:

```
interface I { int x = 0; }
class T1 implements I { int x = 1; }
class T2 extends T1 { int x = 2; }
class T3 extends T2 {
    int x = 3;
    void test() {
        System.out.println("x=\t\t"+x);
        System.out.println("super.x=\t\t"+super.x);
        System.out.println("((T2) this).x=\t"+((T2) this).x);
        System.out.println("((T1) this).x=\t"+((T1) this).x);
        System.out.println("((I) this).x=\t"+((I) this).x);
    }
}
class Test {
    public static void main(String[] args) {
        new T3().test();
    }
}
```

which produces the output:

x=	3	
super.x=		2
((T2) this).x=		2
((T1) this).x=		1
((I) this).x=		0

Within class `T3`, the expression `super.x` is treated exactly as if it were:

```
((T2) this).x
```



```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.11 Method Invocation Expressions

A method invocation expression is used to invoke a class or instance method.

MethodInvocation:

MethodName (*ArgumentListopt*)

Primary . *Identifier* (*ArgumentListopt*)

super . *Identifier* (*ArgumentListopt*)

The definition of *ArgumentList* from §15.8 is repeated here for convenience:

ArgumentList:

Expression

ArgumentList , *Expression*

Resolving a method name at compile time is more complicated than resolving a field name because of the possibility of method overloading. Invoking a method at run time is also more complicated than accessing a field because of the possibility of instance method overriding.

Determining the method that will be invoked by a method invocation expression involves several steps. The following three sections describe the compile-time processing of a method invocation; the determination of the type of the method invocation expression is described in §15.11.3.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.11.1 Compile-Time Step 1: Determine Class or Interface to Search

The first step in processing a method invocation at compile time is to figure out the name of the method to be invoked and which class or interface to check for definitions of methods of that name. There are several cases to consider, depending on the form that precedes the left parenthesis, as follows:

- If the form is *MethodName*, then there are three subcases:
 - If it is a simple name, that is, just an *Identifier*, then the name of the method is the *Identifier* and the class or interface to search is the one whose declaration contains the method invocation.
 - If it is a qualified name of the form *TypeName* . *Identifier*, then the name of the method is the *Identifier* and the class to search is the one named by the *TypeName*. If *TypeName* is the name of an interface rather than a class, then a compile-time error occurs, because this form can invoke only `static` methods and interfaces have no `static` methods.
 - In all other cases, the qualified name has the form *FieldName* . *Identifier*; then the name of the method is the *Identifier* and the class or interface to search is the declared type of the field named by the *FieldName*.
- If the form is *Primary* . *Identifier*, then the name of the method is the *Identifier* and the class or interface to be searched is the type of the *Primary* expression.
- If the form is `super` . *Identifier*, then the name of the method is the *Identifier* and the class to be searched is the superclass of the class whose declaration contains the method invocation. A compile-time error occurs if such a method invocation occurs in an interface, or in the class `Object`, or in a `static` method, a static initializer, or the initializer for a `static` variable. It follows that a method invocation of this form may appear only in a class other than `Object`, and only in the body of an instance method, the body of a constructor, or an initializer for an instance variable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.11.2 Compile-Time Step 2: Determine Method Signature

The hand-writing experts were called upon for their opinion of the signature . . .

--Agatha Christie, *The Mysterious Affair at Styles* (1920), Chapter 11

The second step searches the class or interface determined in the previous step for method declarations. This step uses the name of the method and the types of the argument expressions to locate method declarations that are both *applicable* and *accessible*, that is, declarations that can be correctly invoked on the given arguments. There may be more than one such method declaration, in which case the *most specific* one is chosen. The descriptor (signature plus return type) of the most specific method declaration is one used at run time to do the method dispatch.

15.11.2.1 Find Methods that are Applicable and Accessible

A method declaration is *applicable* to a method invocation if and only if both of the following are true:

- The number of parameters in the method declaration equals the number of argument expressions in the method invocation.
- The type of each actual argument can be converted by method invocation conversion (§5.3) to the type of the corresponding parameter. Method invocation conversion is the same as assignment conversion (§5.2), except that constants of type `int` are never implicitly narrowed to `byte`, `short`, or `char`.

The class or interface determined by the process described in §15.11.1 is searched for all method declarations applicable to this method invocation; method definitions inherited from superclasses and superinterfaces are included in this search.

Whether a method declaration is *accessible* to a method invocation depends on the access modifier (`public`, `none`, `protected`, or `private`) in the method declaration and on where the method invocation appears.

If the class or interface has no method declaration that is both applicable and accessible, then a compile-time error occurs.

In the example program:

```
public class Doubler {
    static int two() { return two(1); }
    private static int two(int i) { return 2*i; }
}

class Test extends Doubler {
    public static long two(long j) {return j+j; }

    public static void main(String[] args) {
        System.out.println(two(3));
        System.out.println(Doubler.two(3)); // compile-time error
    }
}
```

for the method invocation `two(1)` within class `Doubler`, there are two accessible methods named

`two`, but only the second one is applicable, and so that is the one invoked at run time. For the method invocation `two(3)` within class `Test`, there are two applicable methods, but only the one in class `Test` is accessible, and so that is the one to be invoked at run time (the argument `3` is converted to type `long`). For the method invocation `Doubler.two(3)`, the class `Doubler`, not class `Test`, is searched for methods named `two`; the only applicable method is not accessible, and so this method invocation causes a compile-time error.

Another example is:

```
class ColoredPoint {
    int x, y;
    byte color;
    void setColor(byte color) { this.color = color; }
}

class Test {
    public static void main(String[] args) {
        ColoredPoint cp = new ColoredPoint();
        byte color = 37;
        cp.setColor(color);
        cp.setColor(37);
        // compile-time error
    }
}
```

Here, a compile-time error occurs for the second invocation of `setColor`, because no applicable method can be found at compile time. The type of the literal `37` is `int`, and `int` cannot be converted to `byte` by method invocation conversion. Assignment conversion, which is used in the initialization of the variable `color`, performs an implicit conversion of the constant from type `int` to `byte`, which is permitted because the value `37` is small enough to be represented in type `byte`; but such a conversion is not allowed for method invocation conversion.

If the method `setColor` had, however, been declared to take an `int` instead of a `byte`, then both method invocations would be correct; the first invocation would be allowed because method invocation conversion does permit a widening conversion from `byte` to `int`. However, a narrowing cast would then be required in the body of `setColor`:

```
void setColor(int color) { this.color = (byte)color; }
```

15.11.2.2 Choose the Most Specific Method

If more than one method is both accessible and applicable to a method invocation, it is necessary to choose one to provide the descriptor for the run-time method dispatch. Java uses the rule that the *most specific* method is chosen.

The informal intuition is that one method declaration is more specific than another if any invocation handled by the first method could be passed on to the other one without a compile-time type error.

The precise definition is as follows. Let m be a name and suppose that there are two declarations of methods named m , each having n parameters. Suppose that one declaration appears within a class or interface T and that the types of the parameters are T_1, \dots, T_n ; suppose moreover that the other

declaration appears within a class or interface U and that the types of the parameters are U_1, \dots, U_n . Then the method m declared in T is *more specific* than the method m declared in U if and only if both of the following are true:

- T can be converted to U by method invocation conversion.
- T_j can be converted to U_j by method invocation conversion, for all j from 1 to n .

A method is said to be *maximally specific* for a method invocation if it is applicable and accessible and there is no other applicable and accessible method that is more specific.

If there is exactly one maximally specific method, then it is in fact *the most specific* method; it is necessarily more specific than any other method that is applicable and accessible. It is then subjected to some further compile-time checks as described in [§15.11.3](#).

It is possible that no method is the most specific, because there are two or more maximally specific method declarations. In this case, we say that the method invocation is *ambiguous*, and a compile-time error occurs.

15.11.2.3 Example: Overloading Ambiguity

Consider the example:

```
class Point { int x, y; }
class ColoredPoint extends Point { int color; }

class Test {

    static void test(ColoredPoint p, Point q) {
        System.out.println("(ColoredPoint, Point)");
    }

    static void test(Point p, ColoredPoint q) {
        System.out.println("(Point, ColoredPoint)");
    }

    public static void main(String[] args) {
        ColoredPoint cp = new ColoredPoint();
        test(cp, cp);
        // compile-time error
    }
}
```

This example produces an error at compile time. The problem is that there are two declarations of `test` that are applicable and accessible, and neither is more specific than the other. Therefore, the method invocation is ambiguous.

If a third definition of `test` were added:

```
static void test(ColoredPoint p, ColoredPoint q) {
```



```

        System.out.println("(ColoredPoint, ColoredPoint)");
    }

```

then it would be more specific than the other two, and the method invocation would no longer be ambiguous.

15.11.2.4 Example: Return Type Not Considered

As another example, consider:

```

class Point { int x, y; }
class ColoredPoint extends Point { int color; }
class Test {

    static int test(ColoredPoint p) {
        return color;
    }

    static String test(Point p) {
        return "Point";
    }

    public static void main(String[] args) {
        ColoredPoint cp = new ColoredPoint();
        String s = test(cp); // compile-time error
    }
}

```

Here the most specific declaration of method `test` is the one taking a parameter of type `ColoredPoint`. Because the result type of the method is `int`, a compile-time error occurs because an `int` cannot be converted to a `String` by assignment conversion. This example shows that, in Java, the result types of methods do not participate in resolving overloaded methods, so that the second `test` method, which returns a `String`, is not chosen, even though it has a result type that would allow the example program to compile without error.

15.11.2.5 Example: Compile-Time Resolution

The most applicable method is chosen at compile time; its descriptor determines what method is actually executed at run time. If a new method is added to a class, then Java code that was compiled with the old definition of the class might not use the new method, even if a recompilation would cause this method to be chosen.

So, for example, consider two compilation units, one for class `Point`:

```

package points;
public class Point {
    public int x, y;
}

```



```

    public Point(int x, int y) { this.x = x; this.y = y; }
    public String toString() { return toString(""); }

    public String toString(String s) {
        return "(" + x + "," + y + s + " ";
    }
}

```

and one for class ColoredPoint:

```

package points;
public class ColoredPoint extends Point {

    public static final int
        RED = 0, GREEN = 1, BLUE = 2;

    public static String[] COLORS =
        { "red", "green", "blue" };
    public byte color;

    public ColoredPoint(int x, int y, int color) {
        super(x, y); this.color = (byte)color;
    }

    /** Copy all relevant fields of the argument into
        this ColoredPoint object. */
    public void adopt(Point p) { x = p.x; y = p.y; }

    public String toString() {
        String s = "," + COLORS[color];
        return super.toString(s);
    }
}

```

Now consider a third compilation unit that uses ColoredPoint:

```

import points.*;
class Test {
    public static void main(String[] args) {
        ColoredPoint cp =
            new ColoredPoint(6, 6, ColoredPoint.RED);
        ColoredPoint cp2 =
            new ColoredPoint(3, 3, ColoredPoint.GREEN);
        cp.adopt(cp2);
        System.out.println("cp: " + cp);
    }
}

```


The output is:

```
cp: (3,3,red)
```

The application programmer who coded class `Test` has expected to see the word `green`, because the actual argument, a `ColoredPoint`, has a `color` field, and `color` would seem to be a "relevant field" (of course, the documentation for the package `Points` ought to have been much more precise!).

Notice, by the way, that the most specific method (indeed, the only applicable method) for the method invocation of `adopt` has a signature that indicates a method of one parameter, and the parameter is of type `Point`. This signature becomes part of the binary representation of class `Test` produced by the compiler and is used by the method invocation at run time.

Suppose the programmer reported this software error and the maintainer of the `points` package decided, after due deliberation, to correct it by adding a method to class `ColoredPoint`:

```
public void adopt(ColoredPoint p) {  
    adopt((Point)p); color = p.color;  
}
```

If the application programmer then runs the old binary file for `Test` with the new binary file for `ColoredPoint`, the output is still:

```
cp: (3,3,red)
```

because the old binary file for `Test` still has the descriptor "one parameter, whose type is `Point`; void" associated with the method call `cp.adopt(cp2)`. If the source code for `Test` is recompiled, the compiler will then discover that there are now two applicable `adopt` methods, and that the signature for the more specific one is "one parameter, whose type is `ColoredPoint`; void"; running the program will then produce the desired output:

```
cp: (3,3,green)
```

With forethought about such problems, the maintainer of the `points` package could fix the `ColoredPoint` class to work with both newly compiled and old code, by adding defensive code to the old `adopt` method for the sake of old code that still invokes it on `ColoredPoint` arguments:

```
public void adopt(Point p) {  
    if (p instanceof ColoredPoint)  
        color = ((ColoredPoint)p).color;  
    x = p.x; y = p.y;  
}
```


A similar consideration applies if a method is to be moved from a class to a superclass. In this case a forwarding method can be left behind for the sake of old code. The maintainer of the `points` package might choose to move the `adopt` method that takes a `Point` argument up to class `Point`, so that all `Point` objects may enjoy the `adopt` functionality. To avoid compatibility problems with old binary code, the maintainer should leave a forwarding method behind in class `ColoredPoint`:

```
public void adopt(Point p) {  
    if (p instanceof ColoredPoint)  
        color = ((ColoredPoint)p).color;  
    super.adopt(p);  
}
```

Ideally, Java code should be recompiled whenever code that it depends on is changed. However, in an environment where different Java classes are maintained by different organizations, this is not always feasible. Defensive programming with careful attention to the problems of class evolution can make upgraded code much more robust. See [§13](#) for a detailed discussion of binary compatibility and type evolution.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.11.3 Compile-Time Step 3: Is the Chosen Method Appropriate?

If there is a most specific method declaration for a method invocation, it is called the *compile-time declaration* for the method invocation. Two further checks must be made on the compile-time declaration:

- If the method invocation has, before the left parenthesis, a *MethodName* of the form *Identifier*, and the method invocation appears within a `static` method, a static initializer, or the initializer for a `static` variable, then the compile-time declaration must be `static`. If, instead, the compile-time declaration for the method invocation is for an instance method, then a compile-time error occurs. (The reason is that a method invocation of this form cannot be used to invoke an instance method in places where `this` (§15.7.2) is not defined.)
- If the method invocation has, before the left parenthesis, a *MethodName* of the form *TypeName* . *Identifier*, then the compile-time declaration should be `static`. If the compile-time declaration for the method invocation is for an instance method, then a compile-time error occurs. (The reason is that a method invocation of this form does not specify a reference to an object that can serve as `this` within the instance method.)
- If the compile-time declaration for the method invocation is `void`, then the method invocation must be a top-level expression, that is, the *Expression* in an expression statement (§14.7) or in the *ForInit* or *ForUpdate* part of a `for` statement (§14.12), or a compile-time error occurs. (The reason is that such a method invocation produces no value and so must be used only in a situation where a value is not needed.)

The following compile-time information is then associated with the method invocation for use at run time:

- The name of the method.
- The class or interface that contains the compile-time declaration.
- The number of parameters and the types of the parameters, in order.
- The result type, or `void`, as declared in the compile-time declaration.
- The invocation mode, computed as follows:
 - If the compile-time declaration has the `static` modifier, then the invocation mode is `static`.
 - Otherwise, if the compile-time declaration has the `private` modifier, then the invocation mode is `nonvirtual`.
 - Otherwise, if the part of the method invocation before the left parenthesis is of the form `super` . *Identifier*, then the invocation mode is `super`.
 - Otherwise, if the compile-time declaration is in an interface, then the invocation mode is `interface`.
 - Otherwise, the invocation mode is `virtual`.

If the compile-time declaration for the method invocation is not `void`, then the type of the method invocation expression is the result type specified in the compile-time declaration.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.11.4 Runtime Evaluation of Method Invocation

At run time, method invocation requires five steps. First, a *target reference* may be computed. Second, the argument expressions are evaluated. Third, the accessibility of the method to be invoked is checked. Fourth, the actual code for the method to be executed is located. Fifth, a new activation frame is created, synchronization is performed if necessary, and control is transferred to the method code.

15.11.4.1 Compute Target Reference (If Necessary)

There are several cases to consider, depending on which of the three productions for *MethodInvocation* (§15.11) is involved:

- If the first production for *MethodInvocation*, which includes a *MethodName*, is involved, then there are three subcases:
 - If the *MethodName* is a simple name, that is, just an *Identifier*, then there are two subcases:
 - If the invocation mode is `static`, then there is no target reference.
 - Otherwise, the target reference is the value of `this`.
 - If the *MethodName* is a qualified name of the form *TypeName* . *Identifier*, then there is no target reference.
 - If the *MethodName* is a qualified name of the form *FieldName* . *Identifier*, then there are two subcases:
 - If the invocation mode is `static`, then there is no target reference.
 - Otherwise, the target reference is the value of the expression *FieldName*.
- If the second production for *MethodInvocation*, which includes a *Primary*, is involved, then there are two subcases:
 - If the invocation mode is `static`, then there is no target reference. The expression *Primary* is evaluated, but the result is then discarded.
 - Otherwise, the expression *Primary* is evaluated and the result is used as the target reference.

In either case, if the evaluation of the *Primary* expression completes abruptly, then no part of any argument expression appears to have been evaluated, and the method invocation completes abruptly for the same reason.

- If the third production for *MethodInvocation*, which includes the keyword `super`, is involved, then the target reference is the value of `this`.

15.11.4.2 Evaluate Arguments

The argument expressions are evaluated in order, from left to right. If the evaluation of any argument expression completes abruptly, then no part of any argument expression to its right appears to have been evaluated, and the method invocation completes abruptly for the same reason.

15.11.4.3 Check Accessibility of Type and Method

Let *C* be the class containing the method invocation, and let *T* be the class or interface that contained the method being invoked, and *m* be the name of the method, as determined at compile time (§15.11.3).

A Java Virtual Machine must insure, as part of linkage, that the method *m* still exists in the type *T*. If this is not true, then a `NoSuchMethodError` (which is a subclass of `IncompatibleClassChangeError`) occurs. If the invocation mode is `interface`, then the virtual machine must also check that the target reference type still implements the specified interface. If the target reference type does not still implement the interface, then an `IncompatibleClassChangeError` occurs.

The virtual machine must also insure, during linkage, that the type *T* and the method *m* are accessible. For the type *T*:

- If *T* is in the same package as *C*, then *T* is accessible.
- If *T* is in a different package than *C*, and *T* is `public`, then *T* is accessible.

For the method *m*:

- If *m* is `public`, then *m* is accessible. (All members of interfaces are `public` (§9.2)).
- If *m* is `protected`, then *m* is accessible if and only if either *T* is in the same package as *C*, or *C* is *T* or a subclass of *T*.
- If *m* has default (package) access, then *m* is accessible if and only if *T* is in the same package as *C*.
- If *m* is `private`, then *m* is accessible if and only if *C* is *T*.

If either *T* or *m* is not accessible, then an `IllegalAccessError` occurs (§12.3).

15.11.4.4 Locate Method to Invoke

The strategy for method lookup depends on the invocation mode.

If the invocation mode is `static`, no target reference is needed and overriding is not allowed. Method *m* of class *T* is the one to be invoked.

Otherwise, an instance method is to be invoked and there is a target reference. If the target reference is `null`, a `NullPointerException` is thrown at this point. Otherwise, the target reference is said to refer to a *target object* and will be used as the value of the keyword `this` in the invoked method. The other four possibilities for the invocation mode are then considered.

If the invocation mode is `nonvirtual`, overriding is not allowed. Method *m* of class *T* is the one to be invoked.

Otherwise, the invocation mode is `interface`, `virtual`, or `super`, and overriding may occur. A *dynamic method lookup* is used. The dynamic lookup process starts from a class *S*, determined as follows:

- If the invocation mode is `interface` or `virtual`, then *S* is initially the actual run-time class *R* of the target object. If the target object is an array, *R* is the class `Object`. (Note that for invocation mode `interface`, *R* necessarily implements *T*; for invocation mode `virtual`, *R* is necessarily either *T* or a subclass of *T*.)
- If the invocation mode is `super`, then *S* is initially the superclass of the class *C* that contains the method invocation.

The dynamic method lookup uses the following procedure to search class *S*, and then the superclasses of class *S*, as necessary, for method *m*.

- If class *S* contains a declaration for a method named *m* with the same descriptor (same number of parameters, the same parameter types, and the same return type) required by the method invocation as determined at compile time (§15.11.3), then this is the method to be invoked, and the procedure terminates. (We note that as part of the loading and linking process that the virtual machine checks that an overriding method is at least as accessible as the overridden method; an `IncompatibleClassChangeError` occurs if this is not the case.)
- Otherwise, if *S* is not *T*, this same lookup procedure is performed using the superclass of *S*; whatever it comes up with is the result of this lookup.

This procedure will find a suitable method when it reaches class *T*, because otherwise an `IllegalAccessError` would have been thrown by the checks of the previous section §15.11.4.3.

We note that the dynamic lookup process, while described here explicitly, will often be implemented implicitly, for example as a side-effect of the construction and use of per-class method dispatch tables, or the construction of other per-class structures used for efficient dispatch.

15.11.4.5 Create Frame, Synchronize, Transfer Control

A method *m* in some class *S* has been identified as the one to be invoked.

Now a new *activation frame* is created, containing the target reference (if any) and the argument values (if any), as well as enough space for the local variables and stack for the method to be invoked and any other bookkeeping information that may be required by the implementation (stack pointer, program counter, reference to previous activation frame, and the like). If there is not sufficient memory available to create such an activation frame, an `OutOfMemoryError` is thrown.

The newly created activation frame becomes the current activation frame. The effect of this is to assign the argument values to corresponding freshly created parameter variables of the method, and to make the target reference available as `this`, if there is a target reference.

If the method *m* is a `native` method but the necessary native, implementation-dependent binary code has not been loaded (§20.16.13, §20.16.14) or otherwise cannot be dynamically linked, then an `UnsatisfiedLinkError` is thrown.

If the method *m* is not `synchronized`, control is transferred to the body of the method *m* to be invoked.

If the method *m* is `synchronized`, then an object must be locked before the transfer of control. No further progress can be made until the current thread can obtain the lock. If there is a target reference, then the target must be locked; otherwise the `Class` object for class *S*, the class of the method *m*, must be locked. Control is then transferred to the body of the method *m* to be invoked. The object is automatically unlocked when execution of the body of the method has completed, whether normally or abruptly. The locking and unlocking behavior is exactly as if the body of the method were embedded in a `synchronized` statement (§14.17).

15.11.4.6 Implementation Note: Combining Frames

In order to allow certain kinds of code optimization, implementations are permitted some freedom to combine activation frames. Suppose that a method invocation within class *C* is to invoke a method *m* within class *S*. Then the current activation frame may be used to provide space for *S* instead of

creating a new activation frame only if one of the following conditions is true:

- Class *C* and class *S* have the same class loader (§20.14) and class *S* is not `SecurityManager` or a subclass of `SecurityManager`.
- Class *S* has no class loader (this fact indicates that it is a system class); class *S* is not `SecurityManager` or a subclass of `SecurityManager`; and method *m* is known not to call, directly or indirectly, any method of `SecurityManager` (§20.17) or any of its subclasses.

15.11.4.7 Example: Target Reference and Static Methods

When a target reference is computed and then discarded because the invocation mode is `static`, the reference is not examined to see whether it is `null`:

```
class Test {
    static void mountain() {

        System.out.println("Monadnock");

    }

    static Test favorite(){
        System.out.print("Mount ");
        return null;
    }

    public static void main(String[] args) {
        favorite().mountain();
    }
}
```

which prints:

```
Mount Monadnock
```

Here `favorite` returns `null`, yet no `NullPointerException` is thrown.

15.11.4.8 Example: Evaluation Order

As part of an instance method invocation (§15.11), there is an expression that denotes the object to be invoked. This expression appears to be fully evaluated before any part of any argument expression to the method invocation is evaluated.

So, for example, in:

```
class Test {
    public static void main(String[] args) {
        String s = "one";
```



```

        if (s.startsWith(s = "two"))
            System.out.println("oops");
    }
}

```

the occurrence of `s` before `".startsWith"` is evaluated first, before the argument expression `s="two"`. Therefore, a reference to the string `"one"` is remembered as the target reference before the local variable `s` is changed to refer to the string `"two"`. As a result, the `startsWith` method (§20.12.20) is invoked for target object `"one"` with argument `"two"`, so the result of the invocation is `false`, as the string `"one"` does not start with `"two"`. It follows that the test program does not print `"oops"`.

15.11.4.9 Example: Overriding

In the example:

```

class Point {

    final int EDGE = 20;
    int x, y;

    void move(int dx, int dy) {
        x += dx; y += dy;
        if (Math.abs(x) >= EDGE || Math.abs(y) >= EDGE)
            clear();
    }

    void clear() {
        System.out.println("\tPoint clear");
        x = 0; y = 0;
    }
}

class ColoredPoint extends Point {
    int color;

    void clear() {
        System.out.println("\tColoredPoint clear");
        super.clear();
        color = 0;
    }
}

```

the subclass `ColoredPoint` extends the `clear` abstraction defined by its superclass `Point`. It does so by overriding the `clear` method with its own method, which invokes the `clear` method of its superclass, using the form `super.clear`.

This method is then invoked whenever the target object for an invocation of `clear` is a

ColoredPoint. Even the method `move` in `Point` invokes the `clear` method of class `ColoredPoint` when the class of `this` is `ColoredPoint`, as shown by the output of this test program:

```
class Test {
    public static void main(String[] args) {
        Point p = new Point();
        System.out.println("p.move(20,20):");
        p.move(20, 20);
        ColoredPoint cp = new ColoredPoint();
        System.out.println("cp.move(20,20):");
        cp.move(20, 20);
        p = new ColoredPoint();
        System.out.println("p.move(20,20), p colored:");
        p.move(20, 20);
    }
}
```

which is:

```
p.move(20,20):
    Point clear
cp.move(20,20):
    ColoredPoint clear
    Point clear
p.move(20,20), p colored:
    ColoredPoint clear
    Point clear
```

Overriding is sometimes called "late-bound self-reference"; in this example it means that the reference to `clear` in the body of `Point.move` (which is really syntactic shorthand for `this.clear`) invokes a method chosen "late" (at run time, based on the run-time class of the object referenced by `this`) rather than a method chosen "early" (at compile time, based only on the type of `this`). This provides the Java programmer a powerful way of extending abstractions and is a key idea in object-oriented programming.

15.11.4.10 Example: Method Invocation using `super`

An overridden instance method of a superclass may be accessed by using the keyword `super` to access the members of the immediate superclass, bypassing any overriding declaration in the class that contains the method invocation.

When accessing an instance variable, `super` means the same as a cast of `this` ([§15.10.2](#)), but this equivalence does not hold true for method invocation. This is demonstrated by the example:

```
class T1 {
    String s() { return "1"; }
```



```

}

class T2 extends T1 {
    String s() { return "2"; }
}

class T3 extends T2 {
    String s() { return "3"; }

    void test() {
        System.out.println("s()=\t\t"+s());
        System.out.println("super.s()=\t"+super.s());
        System.out.print("(T2)this.s()=\t");
            System.out.println(((T2)this).s());
        System.out.print("(T1)this.s()=\t");
            System.out.println(((T1)this).s());
    }
}

class Test {
    public static void main(String[] args) {
        T3 t3 = new T3();
        t3.test();
    }
}

```

which produces the output:

```

s()=                3
super.s()=          2
((T2)this).s()=      3
((T1)this).s()=      3

```

The casts to types `T1` and `T2` do not change the method that is invoked, because the instance method to be invoked is chosen according to the run-time class of the object referred to be `this`. A cast does not change the class of an object; it only checks that the class is compatible with the specified type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.12 Array Access Expressions

An array access expression refers to a variable that is a component of an array.

ArrayAccess:

ExpressionName [*Expression*]

PrimaryNoNewArray [*Expression*]

An array access expression contains two subexpressions, the *array reference expression* (before the left bracket) and the *index expression* (within the brackets). Note that the array reference expression may be a name or any primary expression that is not an array creation expression ([§15.9](#)).

The type of the array reference expression must be an array type (call it $T[]$, an array whose components are of type T) or a compile-time error results. Then the type of the array access expression is T .

The index expression undergoes unary numeric promotion ([§5.6.1](#)); the promoted type must be `int`.

The result of an array reference is a variable of type T , namely the variable within the array selected by the value of the index expression. This resulting variable, which is a component of the array, is never considered `final`, even if the array reference was obtained from a `final` variable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.12.1 Run-time Evaluation

An array access expression is evaluated using the following procedure:

- First, the array reference expression is evaluated. If this evaluation completes abruptly, then the array access completes abruptly for the same reason and the index expression is not evaluated.
- Otherwise, the index expression is evaluated. If this evaluation completes abruptly, then the array access completes abruptly for the same reason.
- Otherwise, if the value of the array reference expression is `null`, then a `NullPointerException` is thrown.
- Otherwise, the value of the array reference expression indeed refers to an array. If the value of the index expression is less than zero, or greater than or equal to the array's length, then an `IndexOutOfBoundsException` is thrown.
- Otherwise, the result of the array reference is the variable of type *T*, within the array, selected by the value of the index expression. (Note that this resulting variable, which is a component of the array, is never considered `final`, even if the array reference expression is a `final` variable.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.12.2 Examples: Array Access Evaluation Order

In an array access, the expression to the left of the brackets appears to be fully evaluated before any part of the expression within the brackets is evaluated. For example, in the (admittedly monstrous) expression `a[(a=b)[3]]`, the expression `a` is fully evaluated before the expression `(a=b)[3]`; this means that the original value of `a` is fetched and remembered while the expression `(a=b)[3]` is evaluated. This array referenced by the original value of `a` is then subscripted by a value that is element 3 of another array (possibly the same array) that was referenced by `b` and is now also referenced by `a`.

Thus, the example:

```
class Test {
    public static void main(String[] args) {
        int[] a = { 11, 12, 13, 14 };
        int[] b = { 0, 1, 2, 3 };
        System.out.println(a[(a=b)[3]]);
    }
}
```

prints:

14

because the monstrous expression's value is equivalent to `a[b[3]]` or `a[3]` or 14.

If evaluation of the expression to the left of the brackets completes abruptly, no part of the expression within the brackets will appear to have been evaluated. Thus, the example:

```
class Test {
    public static void main(String[] args) {
        int index = 1;
        try {
            skedaddle()[index=2]++;
        } catch (Exception e) {
            System.out.println(e + ", index=" + index);
        }
    }
    static int[] skedaddle() throws Exception {
        throw new Exception("Ciao");
    }
}
```

prints:

java.lang.Exception: Ciao, index=1

because the embedded assignment of 2 to `index` never occurs.

If the array reference expression produces `null` instead of a reference to an array, then a `NullPointerException` is thrown at run time, but only after all parts of the array reference expression have been evaluated and only if these evaluations completed normally. Thus, the example:

```
class Test {  
  
    public static void main(String[] args) {  
        int index = 1;  
        try {  
            nada()[index=2]++;  
        } catch (Exception e) {  
            System.out.println(e + ", index=" + index);  
        }  
    }  
    static int[] nada() { return null; }  
}
```

prints:

```
java.lang.NullPointerException, index=2
```

because the embedded assignment of 2 to `index` occurs before the check for a null pointer. As a related example, the program:

```
class Test {  
  
    public static void main(String[] args) {  
        int[] a = null;  
        try {  
            int i = a[vamoose()];  
            System.out.println(i);  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
  
    static int vamoose() throws Exception {  
        throw new Exception("Twenty-three skidoo!");  
    }  
}
```

always prints:

```
java.lang.Exception: Twenty-three skidoo!
```


A `NullPointerException` never occurs, because the index expression must be completely evaluated before any part of the indexing operation occurs, and that includes the check as to whether the value of the left-hand operand is `null`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.13 Postfix Expressions

Postfix expressions include uses of the postfix ++ and -- operators. Also, as discussed in §15.7, names are not considered to be primary expressions, but are handled separately in the grammar to avoid certain ambiguities. They become interchangeable only here, at the level of precedence of postfix expressions.

PostfixExpression:

Primary

ExpressionName

PostIncrementExpression

PostDecrementExpression

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.13.1 Names

A name occurring in an expression may be, syntactically, an *ExpressionName* (§6.5). The meaning of such an *ExpressionName* depends on its form:

- If it is a simple name, that is, just an *Identifier*, then there are two cases:
 - If the *Identifier* occurs within the scope of a parameter or local variable named by that same *Identifier*, then the type of the *ExpressionName* is the declared type of the parameter or local variable; moreover, the value of the *ExpressionName* is a variable, namely, the parameter or local variable itself.
 - Otherwise, the *ExpressionName* is treated exactly as if it had been the field access expression (§15.10):

`this.Identifier` containing the keyword `this` (§15.7.2).

- Otherwise, if it is a qualified name of the form *PackageName* . *Identifier*, then a compile-time error occurs.
- Otherwise, if it is a qualified name of the form *TypeName* . *Identifier*, then it refers to a `static` field of the class or interface named by the *TypeName*. A compile-time error occurs if *TypeName* does not name a class or interface. A compile-time error occurs if the class or interface named by *TypeName* does not contain an accessible static field named by the *Identifier*. The type of the *ExpressionName* is the declared type of the `static` field. The value of the *ExpressionName* is a variable, namely, the `static` field itself.
- Otherwise, it is a qualified name of the form *Ename* . *Identifier*, where *Ename* is itself an *ExpressionName*, and the *ExpressionName* is treated exactly as if it had been the field access expression (§15.10): `(Ename) . Identifier`

containing a parenthesized expression (§15.7.3).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.13.2 Postfix Increment Operator ++

PostIncrementExpression:

PostfixExpression ++

A postfix expression followed by a ++ operator is a postfix increment expression. The result of the postfix expression must be a variable of a numeric type, or a compile-time error occurs. The type of the postfix increment expression is the type of the variable. The result of the postfix increment expression is not a variable, but a value.

At run time, if evaluation of the operand expression completes abruptly, then the postfix increment expression completes abruptly for the same reason and no incrementation occurs. Otherwise, the value 1 is added to the value of the variable and the sum is stored back into the variable. Before the addition, binary numeric promotion ([§5.6.2](#)) is performed on the value 1 and the value of the variable. If necessary, the sum is narrowed by a narrowing primitive conversion ([§5.1.3](#)) to the type of the variable before it is stored. The value of the postfix increment expression is the value of the variable *before* the new value is stored.

A variable that is declared `final` cannot be incremented, because when an access of a `final` variable is used as an expression, the result is a value, not a variable. Thus, it cannot be used as the operand of a postfix increment operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.13.3 Postfix Decrement Operator --

PostDecrementExpression:

PostfixExpression --

A postfix expression followed by a -- operator is a postfix decrement expression. The result of the postfix expression must be a variable of a numeric type, or a compile-time error occurs. The type of the postfix decrement expression is the type of the variable. The result of the postfix decrement expression is not a variable, but a value.

At run time, if evaluation of the operand expression completes abruptly, then the postfix decrement expression completes abruptly for the same reason and no decrementation occurs. Otherwise, the value 1 is subtracted from the value of the variable and the difference is stored back into the variable. Before the subtraction, binary numeric promotion ([§5.6.2](#)) is performed on the value 1 and the value of the variable. If necessary, the difference is narrowed by a narrowing primitive conversion ([§5.1.3](#)) to the type of the variable before it is stored. The value of the postfix decrement expression is the value of the variable *before* the new value is stored.

A variable that is declared `final` cannot be decremented, because when an access of a `final` variable is used as an expression, the result is a value, not a variable. Thus, it cannot be used as the operand of a postfix decrement operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14 Unary Operators

The *unary operators* include `+`, `-`, `++`, `--`, `~`, `!`, and cast operators. Expressions with unary operators group right-to-left, so that `~~x` means the same as `-(~x)`.

UnaryExpression:

PreIncrementExpression

PreDecrementExpression

+ UnaryExpression

- UnaryExpression

UnaryExpressionNotPlusMinus

PreIncrementExpression:

++ UnaryExpression

PreDecrementExpression:

-- UnaryExpression

UnaryExpressionNotPlusMinus:

PostfixExpression

~ UnaryExpression

! UnaryExpression

CastExpression

The following productions from [§15.15](#) are repeated here for convenience:

CastExpression:

(PrimitiveType) UnaryExpression

(ReferenceType) UnaryExpressionNotPlusMinus

This portion of the Java grammar contains some tricks to avoid two potential syntactic ambiguities.

The first potential ambiguity would arise in expressions such as `(p) + q`, which looks, to a C or C++ programmer, as though it could be either be a cast to type `p` of a unary `+` operating on `q`, or a binary addition of two quantities `p` and `q`. In C and C++, the parser handles this problem by performing a limited amount of semantic analysis as it parses, so that it knows whether `p` is the name of a type or the name of a variable.

Java takes a different approach. The result of the `+` operator must be numeric, and all type names involved in casts on numeric values are known keywords. Thus, if `p` is a keyword naming a primitive type, then `(p)+q` can make sense only as a cast of a unary expression. However, if `p` is not a keyword naming a primitive type, then `(p)+q` can make sense only as a binary arithmetic operation. Similar remarks apply to the `-` operator. The grammar shown above splits *CastExpression* into two cases to make this distinction. The nonterminal *UnaryExpression* includes all unary operator, but the nonterminal *UnaryExpressionNotPlusMinus* excludes uses of all unary operators that could also be binary operators, which in Java are `+` and `-`.

The second potential ambiguity is that the expression `(p)++` could, to a C or C++ programmer, appear to be either a postfix increment of a parenthesized expression or the beginning of a cast, for example, in `(p)++q`. As before, parsers for C and C++ know whether `p` is the name of a type or the name of a variable. But a parser using only one-token lookahead and no semantic analysis during the parse would not be able to tell, when `++` is the lookahead token, whether `(p)` should be considered a *Primary* expression or left alone for later consideration as part of a *CastExpression*.

In Java, the result of the `++` operator must be numeric, and all type names involved in casts on numeric values are known keywords. Thus, if `p` is a keyword naming a primitive type, then `(p)++` can make sense only as a cast of a prefix increment expression, and there had better be an operand such as `q` following the `++`. However, if `p` is not a keyword naming a primitive type, then `(p)++` can make sense only as a postfix increment of `p`. Similar remarks apply to the `--` operator. The nonterminal *UnaryExpressionNotPlusMinus* therefore also excludes uses of the prefix operators `++` and `--`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.1 Prefix Increment Operator ++

A unary expression preceded by a ++ operator is a prefix increment expression. The result of the unary expression must be a variable of a numeric type, or a compile-time error occurs. The type of the prefix increment expression is the type of the variable. The result of the prefix increment expression is not a variable, but a value.

At run time, if evaluation of the operand expression completes abruptly, then the prefix increment expression completes abruptly for the same reason and no incrementation occurs. Otherwise, the value 1 is added to the value of the variable and the sum is stored back into the variable. Before the addition, binary numeric promotion (§5.6.2) is performed on the value 1 and the value of the variable. If necessary, the sum is narrowed by a narrowing primitive conversion (§5.1.3) to the type of the variable before it is stored. The value of the prefix increment expression is the value of the variable *after* the new value is stored.

A variable that is declared `final` cannot be incremented, because when an access of a `final` variable is used as an expression, the result is a value, not a variable. Thus, it cannot be used as the operand of a prefix increment operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.2 Prefix Decrement Operator --

He must increase, but I must decrease.

--John 3:30

A unary expression preceded by a -- operator is a prefix decrement expression. The result of the unary expression must be a variable of a numeric type, or a compile-time error occurs. The type of the prefix decrement expression is the type of the variable. The result of the prefix decrement expression is not a variable, but a value.

At run time, if evaluation of the operand expression completes abruptly, then the prefix decrement expression completes abruptly for the same reason and no decrementation occurs. Otherwise, the value 1 is subtracted from the value of the variable and the difference is stored back into the variable. Before the subtraction, binary numeric promotion (§5.6.2) is performed on the value 1 and the value of the variable. If necessary, the difference is narrowed by a narrowing primitive conversion (§5.1.3) to the type of the variable before it is stored. The value of the prefix decrement expression is the value of the variable *after* the new value is stored.

A variable that is declared `final` cannot be decremented, because when an access of a `final` variable is used as an expression, the result is a value, not a variable. Thus, it cannot be used as the operand of a prefix decrement operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.3 Unary Plus Operator +

The type of the operand expression of the unary + operator must be a primitive numeric type, or a compile-time error occurs. Unary numeric promotion ([§5.6.1](#)) is performed on the operand. The type of the unary plus expression is the promoted type of the operand. The result of the unary plus expression is not a variable, but a value, even if the result of the operand expression is a variable.

At run time, the value of the unary plus expression is the promoted value of the operand.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.4 Unary Minus Operator -

The type of the operand expression of the unary `-` operator must be a primitive numeric type, or a compile-time error occurs. Unary numeric promotion (§5.6.1) is performed on the operand. The type of the unary minus expression is the promoted type of the operand.

At run time, the value of the unary plus expression is the arithmetic negation of the promoted value of the operand.

For integer values, negation is the same as subtraction from zero. Java uses two's-complement representation for integers, and the range of two's-complement values is not symmetric, so negation of the maximum negative `int` or `long` results in that same maximum negative number. Overflow occurs in this case, but no exception is thrown. For all integer values `x`, `-x` equals `(~x) + 1`.

For floating-point values, negation is not the same as subtraction from zero, because if `x` is `+0.0`, then `0.0 - x` equals `+0.0`, but `-x` equals `-0.0`. Unary minus merely inverts the sign of a floating-point number. Special cases of interest:

- If the operand is NaN, the result is NaN (recall that NaN has no sign).
- If the operand is an infinity, the result is the infinity of opposite sign.
- If the operand is a zero, the result is the zero of opposite sign.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.5 Bitwise Complement Operator ~

The type of the operand expression of the unary ~ operator must be a primitive integral type, or a compile-time error occurs. Unary numeric promotion ([§5.6.1](#)) is performed on the operand. The type of the unary bitwise complement expression is the promoted type of the operand.

At run time, the value of the unary bitwise complement expression is the bitwise complement of the promoted value of the operand; note that, in all cases, $\sim x$ equals $(-x) - 1$.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.14.6 Logical Complement Operator !

The type of the operand expression of the unary `!` operator must be `boolean`, or a compile-time error occurs. The type of the unary logical complement expression is `boolean`.

At run time, the value of the unary logical complement expression is `true` if the operand value is `false` and `false` if the operand value is `true`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.15 Cast Expressions

*My days among the dead are passed;
Around me I behold,
Where'er these casual eyes are cast,
The mighty minds of old . . .*
--Robert Southey (1774-1843), *Occasional Pieces*, xviii

A cast expression converts, at run time, a value of one numeric type to a similar value of another numeric type; or confirms, at compile time, that the type of an expression is `boolean`; or checks, at run time, that a reference value refers to an object whose class is compatible with a specified reference type.

CastExpression:

```
( PrimitiveType Dimsopt ) UnaryExpression  
  
( ReferenceType ) UnaryExpressionNotPlusMinus
```

See [§15.14](#) for a discussion of the distinction between *UnaryExpression* and *UnaryExpressionNotPlusMinus*.

The type of a cast expression is the type whose name appears within the parentheses. (The parentheses and the type they contain are sometimes called the *cast operator*.) The result of a cast expression is not a variable, but a value, even if the result of the operand expression is a variable.

At run time, the operand value is converted by casting conversion ([§5.4](#)) to the type specified by the cast operator.

Not all casts are permitted by the Java language. Some casts result in an error at compile time. For example, a primitive value may not be cast to a reference type. Some casts can be proven, at compile time, always to be correct at run time. For example, it is always correct to convert a value of a class type to the type of its superclass; such a cast should require no special action at run time. Finally, some casts cannot be proven to be either always correct or always incorrect at compile time. Such casts require a test at run time. A `ClassCastException` is thrown if a cast is found at run time to be impermissible.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.16 Multiplicative Operators

The operators `*`, `/`, and `%` are called the *multiplicative operators*. They have the same precedence and are syntactically left-associative (they group left-to-right).

MultiplicativeExpression:

UnaryExpression

MultiplicativeExpression `*` *UnaryExpression*

MultiplicativeExpression `/` *UnaryExpression*

MultiplicativeExpression `%` *UnaryExpression*

The type of each of the operands of a multiplicative operator must be a primitive numeric type, or a compile-time error occurs. Binary numeric promotion is performed on the operands ([§5.6.2](#)). The type of a multiplicative expression is the promoted type of its operands. If this promoted type is `int` or `long`, then integer arithmetic is performed; if this promoted type is `float` or `double`, then floating-point arithmetic is performed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.16.1 Multiplication Operator *

Entia non sunt multiplicanda praeter necessitatem.

--William of Occam (c. 1320)

The binary * operator performs multiplication, producing the product of its operands. Multiplication is a commutative operation if the operand expressions have no side effects. While integer multiplication is associative when the operands are all of the same type, floating-point multiplication is not associative.

If an integer multiplication overflows, then the result is the low-order bits of the mathematical product as represented in some sufficiently large two's-complement format. As a result, if overflow occurs, then the sign of the result may not be the same as the sign of the mathematical product of the two operand values.

The result of a floating-point multiplication is governed by the rules of IEEE 754 arithmetic:

- If either operand is NaN, the result is NaN.
- If the result is not NaN, the sign of the result is positive if both operands have the same sign, and negative if the operands have different signs.
- Multiplication of an infinity by a zero results in NaN.
- Multiplication of an infinity by a finite value results in a signed infinity. The sign is determined by the rule stated above.
- In the remaining cases, where neither an infinity or NaN is involved, the product is computed. If the magnitude of the product is too large to represent, we say the operation overflows. The result is then an infinity of appropriate sign. If the magnitude is too small to represent, we say the operation underflows; the result is then a zero of appropriate sign. Otherwise, the product is rounded to the nearest representable value using IEEE 754 round-to-nearest mode. The Java language requires support of gradual underflow as defined by IEEE 754 ([§4.2.4](#)).

Despite the fact that overflow, underflow, or loss of information may occur, evaluation of a multiplication operator * never throws a run-time exception.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.16.2 Division Operator /

Gallia est omnis divisa in partes tres.

--Julius Caesar, *Commentaries on the Gallic Wars* (58 B.C.)

The binary / operator performs division, producing the quotient of its operands. The left-hand operand is the dividend and the right-hand operand is the divisor.

Integer division rounds toward 0. That is, the quotient produced for operands n and d that are integers after binary numeric promotion (§5.6.2) is an integer value q whose magnitude is as large as possible while satisfying {ewc msdncl, EWGraphic, LNG63s 0 /a "langref5ANC3.BMP"}; moreover, q is positive when {ewc msdncl, EWGraphic, LNG63s 1 /a "langref5ANC4.BMP"} and n and d have the same sign, but q is negative when {ewc msdncl, EWGraphic, LNG63s 2 /a "langref5ANC5.BMP"} and n and d have opposite signs. There is one special case that does not satisfy this rule: if the dividend is the negative integer of largest possible magnitude for its type, and the divisor is -1 , then integer overflow occurs and the result is equal to the dividend. Despite the overflow, no exception is thrown in this case. On the other hand, if the value of the divisor in an integer division is 0, then an `ArithmeticException` is thrown.

The result of a floating-point division is determined by the specification of IEEE arithmetic:

- If either operand is NaN, the result is NaN.
- If the result is not NaN, the sign of the result is positive if both operands have the same sign, negative if the operands have different signs.
- Division of an infinity by an infinity results in NaN.
- Division of an infinity by a finite value results in a signed infinity. The sign is determined by the rule stated above.
- Division of a finite value by an infinity results in a signed zero. The sign is determined by the rule stated above.
- Division of a zero by a zero results in NaN; division of zero by any other finite value results in a signed zero. The sign is determined by the rule stated above.
- Division of a nonzero finite value by a zero results in a signed infinity. The sign is determined by the rule stated above.
- In the remaining cases, where neither an infinity, nor a zero, nor NaN is involved, the quotient is computed. If the magnitude of the quotient is too large to represent, we say the operation overflows; the result is then an infinity of appropriate sign. If the magnitude is too small to represent, we say the operation underflows and the result is then a zero of appropriate sign. Otherwise, the quotient is rounded to the nearest representable value using IEEE 754 round-to-nearest mode. The Java language requires support of gradual underflow as defined by IEEE 754 (§4.2.4).

Despite the fact that overflow, underflow, division by zero, or loss of information may occur, evaluation of a floating-point division operator / never throws a run-time exception.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

15.16.3 Remainder Operator %

And on the pedestal these words appear:

"My name is Ozymandias, king of kings:

Look on my works, ye Mighty, and despair!"

Nothing beside remains.

—Percy Bysshe Shelley, *Ozymandias* (1817)

The binary % operator is said to yield the remainder of its operands from an implied division; the left-hand operand is the dividend and the right-hand operand is the divisor.

In C and C++, the remainder operator accepts only integral operands, but in Java, it also accepts floating-point operands.

The remainder operation for operands that are integers after binary numeric promotion (§5.6.2) produces a result value such that $(a/b) * b + (a \% b)$ is equal to a . This identity holds even in the special case that the dividend is the negative integer of largest possible magnitude for its type and the divisor is -1 (the remainder is 0). It follows from this rule that the result of the remainder operation can be negative only if the dividend is negative, and can be positive only if the dividend is positive; moreover, the magnitude of the result is always less than the magnitude of the divisor. If the value of the divisor for an integer remainder operator is 0 , then an `ArithmeticException` is thrown.

Examples:

<code>5%3</code> produces <code>2</code>	(note that <code>5/3</code> produces <code>1</code>)
<code>5%(-3)</code> produces <code>2</code>	(note that <code>5/(-3)</code> produces <code>-1</code>)
<code>(-5)%3</code> produces <code>-2</code>	(note that <code>(-5)/3</code> produces <code>-1</code>)
<code>(-5)%(-3)</code> produces <code>-2</code>	(note that <code>(-5)/(-3)</code> produces <code>1</code>)

The result of a floating-point remainder operation as computed by the % operator is *not* the same as that produced by the remainder operation defined by IEEE 754. The IEEE 754 remainder operation computes the remainder from a rounding division, not a truncating division, and so its behavior is *not* analogous to that of the usual integer remainder operator. Instead, the Java language defines % on floating-point operations to behave in a manner analogous to that of the Java integer remainder operator; this may be compared with the C library function `fmod`. The IEEE 754 remainder operation may be computed by the Java library routine `Math.IEEEremainder` (§20.11.14).

The result of a Java floating-point remainder operation is determined by the rules of IEEE arithmetic:

- If either operand is NaN, the result is NaN.
- If the result is not NaN, the sign of the result equals the sign of the dividend.
- If the dividend is an infinity, or the divisor is a zero, or both, the result is NaN.
- If the dividend is finite and the divisor is an infinity, the result equals the dividend.
- If the dividend is a zero and the divisor is finite, the result equals the dividend.
- In the remaining cases, where neither an infinity, nor a zero, nor NaN is involved, the floating-point remainder r from the division of a dividend n by a divisor d is defined by the mathematical relation $\{ewc msdn cd, EWGraphic, LNG64s 0 /a "langref5ANC6.BMP"\}$ where q is an integer that is negative only if $\{ewc msdn cd, EWGraphic, LNG64s 1 /a "langref5ANC7.BMP"\}$ is negative and positive only if $\{ewc msdn cd, EWGraphic, LNG64s 2 /a "langref5ANC8.BMP"\}$ is positive, and whose magnitude is as large as

possible without exceeding the magnitude of the true mathematical quotient of n and d .

Evaluation of a floating-point remainder operator `%` never throws a run-time exception, even if the right-hand operand is zero. Overflow, underflow, or loss of precision cannot occur.

Examples:

`5.0%3.0` produces `2.0`

`5.0%(-3.0)` produces `2.0`

`(-5.0)%3.0` produces `-2.0`

`(-5.0)%(-3.0)` produces `-2.0`

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

15.17 Additive Operators

The operators `+` and `-` are called the *additive operators*. They have the same precedence and are syntactically left-associative (they group left-to-right).

AdditiveExpression:

MultiplicativeExpression

AdditiveExpression `+` *MultiplicativeExpression*

AdditiveExpression `-` *MultiplicativeExpression*

If the type of either operand of a `+` operator is `String`, then the operation is string concatenation.

Otherwise, the type of each of the operands of the `+` operator must be a primitive numeric type, or a compile-time error occurs.

In every case, the type of each of the operands of the binary `-` operator must be a primitive numeric type, or a compile-time error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.17.1 String Concatenation Operator +

If only one operand expression is of type `String`, then string conversion is performed on the other operand to produce a string at run time. The result is a reference to a newly created `String` object that is the concatenation of the two operand strings. The characters of the left-hand operand precede the characters of the right-hand operand in the newly created string.

15.17.1.1 String Conversion

Any type may be converted to type `String` by *string conversion*.

A value x of primitive type T is first converted to a reference value as if by giving it as an argument to an appropriate class instance creation expression:

- If T is `boolean`, then use `new Boolean(x)` (§20.4).
- If T is `char`, then use `new Character(x)` (§20.5).
- If T is `byte`, `short`, or `int`, then use `new Integer(x)` (§20.7).
- If T is `long`, then use `new Long(x)` (§20.8).
- If T is `float`, then use `new Float(x)` (§20.9).
- If T is `double`, then use `new Double(x)` (§20.10).

This reference value is then converted to type `String` by string conversion.

Now only reference values need to be considered. If the reference is `null`, it is converted to the string `"null"` (four ASCII characters `n`, `u`, `l`, `l`). Otherwise, the conversion is performed as if by an invocation of the `toString` method of the referenced object with no arguments; but if the result of invoking the `toString` method is `null`, then the string `"null"` is used instead. The `toString` method (§20.1.2) is defined by the primordial class `Object` (§20.1); many classes override it, notably `Boolean`, `Character`, `Integer`, `Long`, `Float`, `Double`, and `String`.

15.17.1.2 Optimization of String Concatenation

An implementation may choose to perform conversion and concatenation in one step to avoid creating and then discarding an intermediate `String` object. To increase the performance of repeated string concatenation, a Java compiler may use the `StringBuffer` class (§20.13) or a similar technique to reduce the number of intermediate `String` objects that are created by evaluation of an expression.

For primitive objects, an implementation may also optimize away the creation of a wrapper object by converting directly from a primitive type to a string.

15.17.1.3 Examples of String Concatenation

The example expression:

```
"The square root of 2 is " + Math.sqrt(2)
```


produces the result:

```
"The square root of 2 is 1.4142135623730952"
```

The + operator is syntactically left-associative, no matter whether it is later determined by type analysis to represent string concatenation or addition. In some cases care is required to get the desired result. For example, the expression:

```
a + b + c
```

is always regarded as meaning:

```
(a + b) + c
```

Therefore the result of the expression:

```
1 + 2 + " fiddlers"
```

is:

```
"3 fiddlers"
```

but the result of:

```
"fiddlers " + 1 + 2
```

is:

```
"fiddlers 12"
```

In this jocular little example:

```
class Bottles {  
    static void printSong(Object stuff, int n) {  
        String plural = "s";  
        loop: while (true) {  
            System.out.println(n + " bottle" + plural  
                + " of " + stuff + " on the wall,");  
            System.out.println(n + " bottle" + plural  
                + " of " + stuff + ";");  
            System.out.println("You take one down "  
                + "and pass it around:");  
        }  
    }  
}
```



```

        --n;
        plural = (n == 1) ? "" : "s";
        if (n == 0)
            break loop;
        System.out.println(n + " bottle" + plural
            + " of " + stuff + " on the wall!");
        System.out.println();
    }
    System.out.println("No bottles of " +
        stuff + " on the wall!");
}
}

```

the method `printSong` will print a version of a children's song. Popular values for `stuff` include "pop" and "beer"; the most popular value for `n` is 100. Here is the output that results from `Bottles.printSong("slime", 3)`:

```

3 bottles of slime on the wall,
3 bottles of slime;
You take one down and pass it around:
2 bottles of slime on the wall!

2 bottles of slime on the wall,
2 bottles of slime;
You take one down and pass it around:
1 bottle of slime on the wall!

1 bottle of slime on the wall,
1 bottle of slime;
You take one down and pass it around:
No bottles of slime on the wall!

```

In the code, note the careful conditional generation of the singular "bottle" when appropriate rather than the plural "bottles"; note also how the string concatenation operator was used to break the long constant string:

```
"You take one down and pass it around:"
```

into two pieces to avoid an inconveniently long line in the source code.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.17.2 Additive Operators (+ and -) for Numeric Types

The binary `+` operator performs addition when applied to two operands of numeric type, producing the sum of the operands. The binary `-` operator performs subtraction, producing the difference of two numeric operands.

Binary numeric promotion is performed on the operands (§5.6.2). The type of an additive expression on numeric operands is the promoted type of its operands. If this promoted type is `int` or `long`, then integer arithmetic is performed; if this promoted type is `float` or `double`, then floating-point arithmetic is performed.

Addition is a commutative operation if the operand expressions have no side effects. Integer addition is associative when the operands are all of the same type, but floating-point addition is not associative.

If an integer addition overflows, then the result is the low-order bits of the mathematical sum as represented in some sufficiently large two's-complement format. If overflow occurs, then the sign of the result is not the same as the sign of the mathematical sum of the two operand values.

The result of a floating-point addition is determined using the following rules of IEEE arithmetic:

- If either operand is NaN, the result is NaN.
- The sum of two infinities of opposite sign is NaN.
- The sum of two infinities of the same sign is the infinity of that sign.
- The sum of an infinity and a finite value is equal to the infinite operand.
- The sum of two zeros of opposite sign is positive zero.
- The sum of two zeros of the same sign is the zero of that sign.
- The sum of a zero and a nonzero finite value is equal to the nonzero operand.
- The sum of two nonzero finite values of the same magnitude and opposite sign is positive zero.
- In the remaining cases, where neither an infinity, nor a zero, nor NaN is involved, and the operands have the same sign or have different magnitudes, the sum is computed. If the magnitude of the sum is too large to represent, we say the operation overflows; the result is then an infinity of appropriate sign. If the magnitude is too small to represent, we say the operation underflows; the result is then a zero of appropriate sign. Otherwise, the sum is rounded to the nearest representable value using IEEE 754 round-to-nearest mode. The Java language requires support of gradual underflow as defined by IEEE 754 (§4.2.4).

The binary `-` operator performs subtraction when applied to two operands of numeric type producing the difference of its operands; the left-hand operand is the minuend and the right-hand operand is the subtrahend. For both integer and floating-point subtraction, it is always the case that $a - b$ produces the same result as $a + (-b)$. Note that, for integer values, subtraction from zero is the same as negation. However, for floating-point operands, subtraction from zero is *not* the same as negation, because if x is $+0.0$, then $0.0 - x$ equals $+0.0$, but $-x$ equals -0.0 .

Despite the fact that overflow, underflow, or loss of information may occur, evaluation of a numeric additive operator never throws a run-time exception.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.18 Shift Operators

What, I say, is to become of those wretches?

... What more can you say to them than "shift for yourselves?"

--Thomas Paine, *The American Crisis* (1780)

The *shift operators* include left shift `<<`, signed right shift `>>`, and unsigned right shift `>>>`; they are syntactically left-associative (they group left-to-right). The left-hand operand of a shift operator is the value to be shifted; the right-hand operand specifies the shift distance.

ShiftExpression:

AdditiveExpression

ShiftExpression `<<` *AdditiveExpression*

ShiftExpression `>>` *AdditiveExpression*

ShiftExpression `>>>` *AdditiveExpression*

The type of each of the operands of a shift operator must be a primitive integral type, or a compile-time error occurs. Binary numeric promotion (§5.6.2) is *not* performed on the operands; rather, unary numeric promotion (§5.6.1) is performed on each operand separately. The type of the shift expression is the promoted type of the left-hand operand.

If the promoted type of the left-hand operand is `int`, only the five lowest-order bits of the right-hand operand are used as the shift distance. It is as if the right-hand operand were subjected to a bitwise logical AND operator `&` (§15.21.1) with the mask value `0x1f`. The shift distance actually used is therefore always in the range 0 to 31, inclusive.

If the promoted type of the left-hand operand is `long`, then only the six lowest-order bits of the right-hand operand are used as the shift distance. It is as if the right-hand operand were subjected to a bitwise logical AND operator `&` (§15.21.1) with the mask value `0x3f`. The shift distance actually used is therefore always in the range 0 to 63, inclusive.

At run time, shift operations are performed on the two's complement integer representation of the value of the left operand.

The value of `n<<s` is `n` left-shifted `s` bit positions; this is equivalent (even if overflow occurs) to multiplication by two to the power `s`.

The value of `n>>s` is `n` right-shifted `s` bit positions with sign-extension. The resulting value is {ewc msdn cd, EWGraphic, LNG71s 0 /a "langref5ANC9.BMP"}. For nonnegative values of `n`, this is equivalent to truncating integer division, as computed by the integer division operator `/`, by two to the power `s`.

The value of `n>>>s` is `n` right-shifted `s` bit positions with zero-extension. If `n` is positive, then the result is the same as that of `n>>s`; if `n` is negative, the result is equal to that of the expression `(n>>s) + (2<<~s)` if the type of the left-hand operand is `int`, and to the result of the expression `(n>>s) + (2L<<~s)` if the type of the left-hand operand is `long`. The added term `(2<<~s)` or `(2L<<~s)` cancels out the propagated sign bit. (Note that, because of the implicit masking of the right-hand operand of a shift operator, `~s` as a shift distance is equivalent to `31-s` when shifting an

int value and to 63-s when shifting a long value.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.19 Relational Operators

The *relational operators* are syntactically left-associative (they group left-to-right), but this fact is not useful; for example, `a<b<c` parses as `(a<b)<c`, which is always a compile-time error, because the type of `a<b` is always `boolean` and `<` is not an operator on `boolean` values.

RelationalExpression:

ShiftExpression

RelationalExpression `<` *ShiftExpression*

RelationalExpression `>` *ShiftExpression*

RelationalExpression `<=` *ShiftExpression*

RelationalExpression `>=` *ShiftExpression*

RelationalExpression *instanceof* *ReferenceType*

The type of a relational expression is always `boolean`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.19.1 Numerical Comparison Operators <, <=, >, and >=

The type of each of the operands of a numerical comparison operator must be a primitive numeric type, or a compile-time error occurs. Binary numeric promotion is performed on the operands (§5.6.2). If the promoted type of the operands is `int` or `long`, then signed integer comparison is performed; if this promoted type is `float` or `double`, then floating-point comparison is performed.

The result of a floating-point comparison, as determined by the specification of the IEEE 754 standard, is:

- If either operand is NaN, then the result is `false`.
- All values other than NaN are ordered, with negative infinity less than all finite values, and positive infinity greater than all finite values.
- Positive zero and negative zero are considered equal. Therefore, `-0.0 < 0.0` is `false`, for example, but `-0.0 <= 0.0` is `true`. (Note, however, that the methods `Math.min` (§20.11.27, §20.11.28) and `Math.max` (§20.11.31, §20.11.32) treat negative zero as being strictly smaller than positive zero.)

Subject to these considerations for floating-point numbers, the following rules then hold for integer operands or for floating-point operands other than NaN:

- The value produced by the `<` operator is `true` if the value of the left-hand operand is less than the value of the right-hand operand, and otherwise is `false`.
- The value produced by the `<=` operator is `true` if the value of the left-hand operand is less than or equal to the value of the right-hand operand, and otherwise is `false`.
- The value produced by the `>` operator is `true` if the value of the left-hand operand is greater than the value of the right-hand operand, and otherwise is `false`.
- The value produced by the `>=` operator is `true` if the value of the left-hand operand is greater than or equal to the value of the right-hand operand, and otherwise is `false`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.19.2 Type Comparison Operator instanceof

The type of a *RelationalExpression* operand of the `instanceof` operator must be a reference type or the null type; otherwise, a compile-time error occurs. The *ReferenceType* mentioned after the `instanceof` operator must denote a reference type; otherwise, a compile-time error occurs.

At run time, the result of the `instanceof` operator is `true` if the value of the *RelationalExpression* is not `null` and the reference could be cast (§15.15) to the *ReferenceType* without raising a `ClassCastException`. Otherwise the result is `false`.

If a cast of the *RelationalExpression* to the *ReferenceType* would be rejected as a compile-time error, then the `instanceof` relational expression likewise produces a compile-time error. In such a situation, the result of the `instanceof` expression could never be `true`.

Consider the example program:

```
class Point { int x, y; }
class Element { int atomicNumber; }
class Test {
    public static void main(String[] args) {
        Point p = new Point();
        Element e = new Element();
        if (e instanceof Point) { // compile-time error
            System.out.println("I get your point!");
            p = (Point)e; // compile-time error
        }
    }
}
```

This example results in two compile-time errors. The cast `(Point)e` is incorrect because no instance of `Element` or any of its possible subclasses (none are shown here) could possibly be an instance of any subclass of `Point`. The `instanceof` expression is incorrect for exactly the same reason. If, on the other hand, the class `Point` were a subclass of `Element` (an admittedly strange notion in this example):

```
class Point extends Element { int x, y; }
```

then the cast would be possible, though it would require a run-time check, and the `instanceof` expression would then be sensible and valid. The cast `(Point)e` would never raise an exception because it would not be executed if the value of `e` could not correctly be cast to type `Point`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.20 Equality Operators

The equality operators are syntactically left-associative (they group left-to-right), but this fact is essentially never useful; for example, `a==b==c` parses as `(a==b)==c`. The result type of `a==b` is always `boolean`, and `c` must therefore be of type `boolean` or a compile-time error occurs. Thus, `a==b==c` does *not* test to see whether `a`, `b`, and `c` are all equal.

EqualityExpression:

RelationalExpression

EqualityExpression `==` *RelationalExpression*

EqualityExpression `!=` *RelationalExpression*

The `==` (equal to) and the `!=` (not equal to) operators are analogous to the relational operators except for their lower precedence. Thus, `a<b==c<d` is `true` whenever `a<b` and `c<d` have the same truth value.

The equality operators may be used to compare two operands of numeric type, or two operands of type `boolean`, or two operands that are each of either reference type or the null type. All other cases result in a compile-time error. The type of an equality expression is always `boolean`.

In all cases, `a!=b` produces the same result as `!(a==b)`. The equality operators are commutative if the operand expressions have no side effects.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.20.1 Numerical Equality Operators == and !=

If the operands of an equality operator are both of primitive numeric type, binary numeric promotion is performed on the operands (§5.6.2). If the promoted type of the operands is `int` or `long`, then an integer equality test is performed; if the promoted type is `float` or `double`, then a floating-point equality test is performed.

Floating-point equality testing is performed in accordance with the rules of the IEEE 754 standard:

- If either operand is NaN, then the result of `==` is `false` but the result of `!=` is `true`. Indeed, the test `x!=x` is true if and only if the value of `x` is NaN. (The methods `Float.isNaN` (§20.9.19) and `Double.isNaN` (§20.10.17) may also be used to test whether a value is NaN.)
- Positive zero and negative zero are considered equal. Therefore, `-0.0==0.0` is `true`, for example.
- Otherwise, two distinct floating-point values are considered unequal by the equality operators. In particular, there is one value representing positive infinity and one value representing negative infinity; each compares equal only to itself, and each compares unequal to all other values.

Subject to these considerations for floating-point numbers, the following rules then hold for integer operands or for floating-point operands other than NaN:

- The value produced by the `==` operator is `true` if the value of the left-hand operand is equal to the value of the right-hand operand; otherwise, the result is `false`.
- The value produced by the `!=` operator is `true` if the value of the left-hand operand is not equal to the value of the right-hand operand; otherwise, the result is `false`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.20.2 Boolean Equality Operators == and !=

If the operands of an equality operator are both of type `boolean`, then the operation is boolean equality. The `boolean` equality operators are associative.

The result of `==` is `true` if the operands are both `true` or both `false`; otherwise, the result is `false`.

The result of `!=` is `false` if the operands are both `true` or both `false`; otherwise, the result is `true`. Thus `!=` behaves the same as `^` ([§15.21.2](#)) when applied to boolean operands.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.20.3 Reference Equality Operators == and !=

Things are more like they are now than they ever were before.

--Dwight D. Eisenhower

If the operands of an equality operator are both of either reference type or the null type, then the operation is object equality.

A compile-time error occurs if it is impossible to convert the type of either operand to the type of the other by a casting conversion ([§5.4](#)). The run-time values of the two operands would necessarily be unequal.

At run time, the result of `==` is `true` if the operand values are both `null` or both refer to the same object or array; otherwise, the result is `false`.

The result of `!=` is `false` if the operand values are both `null` or both refer to the same object or array; otherwise, the result is `true`.

While `==` may be used to compare references of type `String`, such an equality test determines whether or not the two operands refer to the same `String` object. The result is `false` if the operands are distinct `String` objects, even if they contain the same sequence of characters. The contents of two strings `s` and `t` can be tested for equality by the method invocation `s.equals(t)` ([§20.12.9](#)). See also [§3.10.5](#) and [§20.12.47](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.21 Bitwise and Logical Operators

The *bitwise operators* and *logical operators* include the AND operator `&`, exclusive OR operator `^`, and inclusive OR operator `|`. These operators have different precedence, with `&` having the highest precedence and `|` the lowest precedence. Each of these operators is syntactically left-associative (each groups left-to-right). Each operator is commutative if the operand expressions have no side effects. Each operator is associative.

AndExpression:

EqualityExpression

AndExpression `&` *EqualityExpression*

ExclusiveOrExpression:

AndExpression

ExclusiveOrExpression `^` *AndExpression*

InclusiveOrExpression:

ExclusiveOrExpression

InclusiveOrExpression `|` *ExclusiveOrExpression*

The bitwise and logical operators may be used to compare two operands of numeric type or two operands of type `boolean`. All other cases result in a compile-time error.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.21.1 Integer Bitwise Operators &, ^, and |

When both operands of an operator &, ^, or | are of primitive integral type, binary numeric promotion is first performed on the operands (\$5.6.2). The type of the bitwise operator expression is the promoted type of the operands.

For &, the result value is the bitwise AND of the operand values.

For ^, the result value is the bitwise exclusive OR of the operand values.

For |, the result value is the bitwise inclusive OR of the operand values.

For example, the result of the expression `0xff00 & 0xf0f0` is `0xf000`. The result of `0xff00 ^ 0xf0f0` is `0x0ff0`. The result of `0xff00 | 0xf0f0` is `0xffff0`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.21.2 Boolean Logical Operators &, ^, and |

When both operands of a &, ^, or | operator are of type `boolean`, then the type of the bitwise operator expression is `boolean`.

For &, the result value is `true` if both operand values are `true`; otherwise, the result is `false`.

For ^, the result value is `true` if the operand values are different; otherwise, the result is `false`.

For |, the result value is `false` if both operand values are `false`; otherwise, the result is `true`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.22 Conditional-And Operator &&

The `&&` operator is like `&` (§15.21.2), but evaluates its right-hand operand only if the value of its left-hand operand is `true`. It is syntactically left-associative (it groups left-to-right). It is fully associative with respect to both side effects and result value; that is, for any expressions *a*, *b*, and *c*, evaluation of the expression `((a) && (b)) && (c)` produces the same result, with the same side effects occurring in the same order, as evaluation of the expression `(a) && ((b) && (c))`.

ConditionalAndExpression:

InclusiveOrExpression

ConditionalAndExpression && *InclusiveOrExpression*

Each operand of `&&` must be of type `boolean`, or a compile-time error occurs. The type of a conditional-and expression is always `boolean`.

At run time, the left-hand operand expression is evaluated first; if its value is `false`, the value of the conditional-and expression is `false` and the right-hand operand expression is not evaluated. If the value of the left-hand operand is `true`, then the right-hand expression is evaluated and its value becomes the value of the conditional-and expression. Thus, `&&` computes the same result as `&` on `boolean` operands. It differs only in that the right-hand operand expression is evaluated conditionally rather than always.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

15.23 Conditional-Or Operator ||

The `||` operator is like `|` (§15.21.2), but evaluates its right-hand operand only if the value of its left-hand operand is `false`. It is syntactically left-associative (it groups left-to-right). It is fully associative with respect to both side effects and result value; that is, for any expressions *a*, *b*, and *c*, evaluation of the expression `(a) || (b) || (c)` produces the same result, with the same side effects occurring in the same order, as evaluation of the expression `(a) || ((b) || (c))`.

ConditionalOrExpression:

ConditionalAndExpression

ConditionalOrExpression `||` *ConditionalAndExpression*

Each operand of `||` must be of type `boolean`, or a compile-time error occurs. The type of a conditional-or expression is always `boolean`.

At run time, the left-hand operand expression is evaluated first; if its value is `true`, the value of the conditional-or expression is `true` and the right-hand operand expression is not evaluated. If the value of the left-hand operand is `false`, then the right-hand expression is evaluated and its value becomes the value of the conditional-or expression. Thus, `||` computes the same result as `|` on `boolean` operands. It differs only in that the right-hand operand expression is evaluated conditionally rather than always.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.24 Conditional Operator ? :

The conditional operator `?` : uses the boolean value of one expression to decide which of two other expressions should be evaluated.

The conditional operator is syntactically right-associative (it groups right-to-left), so that `a?b:c?d:e?f:g` means the same as `a?b:(c?d:(e?f:g))`.

ConditionalExpression:

ConditionalOrExpression

ConditionalOrExpression ? Expression : ConditionalExpression

The conditional operator has three operand expressions; `?` appears between the first and second expressions, and `:` appears between the second and third expressions.

The first expression must be of type `boolean`, or a compile-time error occurs.

The conditional operator may be used to choose between second and third operands of numeric type, or second and third operands of type `boolean`, or second and third operands that are each of either reference type or the null type. All other cases result in a compile-time error.

Note that it is not permitted for either the second or the third operand expression to be an invocation of a `void` method. In fact, it is not permitted for a conditional expression to appear in any context where an invocation of a `void` method could appear ([§14.7](#)).

The type of a conditional expression is determined as follows:

- If the second and third operands have the same type (which may be the null type), then that is the type of the conditional expression.
- Otherwise, if the second and third operands have numeric type, then there are several cases:
 - If one of the operands is of type `byte` and the other is of type `short`, then the type of the conditional expression is `short`.
 - If one of the operands is of type `T` where `T` is `byte`, `short`, or `char`, and the other operand is a constant expression of type `int` whose value is representable in type `T`, then the type of the conditional expression is `T`.
 - Otherwise, binary numeric promotion ([§5.6.2](#)) is applied to the operand types, and the type of the conditional expression is the promoted type of the second and third operands.
- If one of the second and third operands is of the null type and the type of the other is a reference type, then the type of the conditional expression is that reference type.
- If the second and third operands are of different reference types, then it must be possible to convert one of the types to the other type (call this latter type `T`) by assignment conversion ([§5.2](#)); the type of the conditional expression is `T`. It is a compile-time error if neither type is assignment compatible with the other type.

At run time, the first operand expression of the conditional expression is evaluated first; its `boolean` value is then used to choose either the second or the third operand expression:

- If the value of the first operand is `true`, then the second operand expression is chosen.
- If the value of the first operand is `false`, then the third operand expression is chosen.

The chosen operand expression is then evaluated and the resulting value is converted to the type of the conditional expression as determined by the rules stated above. The operand expression not chosen is not evaluated for that particular evaluation of the conditional expression.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.25 Assignment Operators

There are 12 *assignment operators*; all are syntactically right-associative (they group right-to-left). Thus, `a=b=c` means `a=(b=c)`, which assigns the value of `c` to `b` and then assigns the value of `b` to `a`.

AssignmentExpression:

ConditionalExpression

Assignment

Assignment:

LeftHandSide AssignmentOperator AssignmentExpression

LeftHandSide:

ExpressionName

FieldAccess

ArrayAccess

AssignmentOperator: one of

`= *= /= %= += -= <=> >>= &= ^= |=`

The result of the first operand of an assignment operator must be a variable, or a compile-time error occurs. This operand may be a named variable, such as a local variable or a field of the current object or class, or it may be a computed variable, as can result from a field access ([§15.10](#)) or an array access ([§15.12](#)). The type of the assignment expression is the type of the variable.

At run time, the result of the assignment expression is the value of the variable after the assignment has occurred. The result of an assignment expression is not itself a variable.

A variable that is declared `final` cannot be assigned to, because when an access of a `final` variable is used as an expression, the result is a value, not a variable, and so it cannot be used as the operand of an assignment operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.25.1 Simple Assignment Operator =

A compile-time error occurs if the type of the right-hand operand cannot be converted to the type of the variable by assignment conversion (§5.2).

At run time, the expression is evaluated in one of two ways. If the left-hand operand expression is not an array access expression, then three steps are required:

- First, the left-hand operand is evaluated to produce a variable. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason; the right-hand operand is not evaluated and no assignment occurs.
- Otherwise, the right-hand operand is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and no assignment occurs.
- Otherwise, the value of the right-hand operand is converted to the type of the left-hand variable and the result of the conversion is stored into the variable.

If the left-hand operand expression is an array access expression (§15.12), then many steps are required:

- First, the array reference subexpression of the left-hand operand array access expression is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason; the index subexpression (of the left-hand operand array access expression) and the right-hand operand are not evaluated and no assignment occurs.
- Otherwise, the index subexpression of the left-hand operand array access expression is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and the right-hand operand is not evaluated and no assignment occurs.
- Otherwise, the right-hand operand is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and no assignment occurs.
- Otherwise, if the value of the array reference subexpression is `null`, then no assignment occurs and a `NullPointerException` is thrown.
- Otherwise, the value of the array reference subexpression indeed refers to an array. If the value of the index subexpression is less than zero, or greater than or equal to the length of the array, then no assignment occurs and an `IndexOutOfBoundsException` is thrown.
- Otherwise, the value of the index subexpression is used to select a component of the array referred to by the value of the array reference subexpression. This component is a variable; call its type *SC*. Also, let *TC* be the type of the left-hand operand of the assignment operator as determined at compile time.
 - If *TC* is a primitive type, then *SC* is necessarily the same as *TC*. The value of the right-hand operand is converted to a value of type *TC* and stored into the selected array component.
 - If *T* is a reference type, then *SC* may not be the same as *T*, but rather a type that extends or implements *TC*. Let *RC* be the class of the object referred to by the value of the right-hand operand at run time. The compiler may be able to prove at compile time that the array component will be of type *TC* exactly (for example, *TC* might be `final`). But if the compiler cannot prove at compile time that the array component will be of type *TC* exactly, then a check must be performed at run time to ensure that the class *RC* is assignment compatible (§5.2) with the actual type *SC* of the array component. This check is similar to a narrowing cast (§5.4, §15.15), except that if the check fails, an `ArrayStoreException` is thrown rather than a `ClassCastException`. Therefore:
 - If class *RC* is not assignable to type *SC*, then no assignment occurs and an `ArrayStoreException` is thrown.

- Otherwise, the reference value of the right-hand operand is stored into the selected array component.

The rules for assignment to an array component are illustrated by the following example program:

```
class ArrayReferenceThrow extends RuntimeException { }

class IndexThrow extends RuntimeException { }

class RightHandSideThrow extends RuntimeException { }

class IllustrateSimpleArrayAssignment {

    static Object[] objects = { new Object(), new Object() };

    static Thread[] threads = { new Thread(), new Thread() };

    static Object[] arrayThrow() {
        throw new ArrayReferenceThrow();
    }

    static int indexThrow() { throw new IndexThrow(); }

    static Thread rightThrow() {
        throw new RightHandSideThrow();
    }

    static String name(Object q) {
        String sq = q.getClass().getName();
        int k = sq.lastIndexOf('.');
        return (k < 0) ? sq : sq.substring(k+1);
    }

    static void testFour(Object[] x, int j, Object y) {
        String sx = x == null ? "null" : name(x[0]) + "s";
        String sy = name(y);
        System.out.println();
        try {
            System.out.print(sx + "[throw]=throw => ");
        }
    }
}
```



```

        x[indexThrow()] = rightThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sx + "[throw]=" + sy + " => ");
        x[indexThrow()] = y;
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sx + "[" + j + "]=throw => ");
        x[j] = rightThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sx + "[" + j + "]=" + sy + " => ");
        x[j] = y;
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
}

public static void main(String[] args) {
    try {
        System.out.print("throw[throw]=throw => ");
        arrayThrow()[indexThrow()] = rightThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[throw]=Thread => ");
        arrayThrow()[indexThrow()] = new Thread();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]=throw => ");
        arrayThrow()[1] = rightThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]=Thread => ");
        arrayThrow()[1] = new Thread();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }

    testFour(null, 1, new StringBuffer());
    testFour(null, 1, new StringBuffer());
    testFour(null, 9, new Thread());
    testFour(null, 9, new Thread());
    testFour(objects, 1, new StringBuffer());
    testFour(objects, 1, new Thread());
    testFour(objects, 9, new StringBuffer());
    testFour(objects, 9, new Thread());
    testFour(threads, 1, new StringBuffer());
    testFour(threads, 1, new Thread());
    testFour(threads, 9, new StringBuffer());
    testFour(threads, 9, new Thread());
}

```



```
}
```

This program prints:

```
throw[throw]=throw => ArrayReferenceThrow  
throw[throw]=Thread => ArrayReferenceThrow  
throw[1]=throw => ArrayReferenceThrow  
throw[1]=Thread => ArrayReferenceThrow
```

```
null[throw]=throw => IndexThrow  
null[throw]=StringBuffer => IndexThrow  
null[1]=throw => RightHandSideThrow  
null[1]=StringBuffer => NullPointerException
```

```
null[throw]=throw => IndexThrow  
null[throw]=StringBuffer => IndexThrow  
null[1]=throw => RightHandSideThrow  
null[1]=StringBuffer => NullPointerException
```

```
null[throw]=throw => IndexThrow  
null[throw]=Thread => IndexThrow  
null[9]=throw => RightHandSideThrow  
null[9]=Thread => NullPointerException
```

```
null[throw]=throw => IndexThrow  
null[throw]=Thread => IndexThrow  
null[9]=throw => RightHandSideThrow  
null[9]=Thread => NullPointerException
```

```
Objects[throw]=throw => IndexThrow  
Objects[throw]=StringBuffer => IndexThrow  
Objects[1]=throw => RightHandSideThrow  
Objects[1]=StringBuffer => Okay!
```

```
Objects[throw]=throw => IndexThrow  
Objects[throw]=Thread => IndexThrow  
Objects[1]=throw => RightHandSideThrow  
Objects[1]=Thread => Okay!
```

```
Objects[throw]=throw => IndexThrow  
Objects[throw]=StringBuffer => IndexThrow  
Objects[9]=throw => RightHandSideThrow  
Objects[9]=StringBuffer => IndexOutOfBoundsException
```

```
Objects[throw]=throw => IndexThrow  
Objects[throw]=Thread => IndexThrow
```



```
Objects[9]=throw => RightHandSideThrow  
Objects[9]=Thread => IndexOutOfBoundsException
```

```
Threads[throw]=throw => IndexThrow  
Threads[throw]=StringBuffer => IndexThrow  
Threads[1]=throw => RightHandSideThrow  
Threads[1]=StringBuffer => ArrayStoreException
```

```
Threads[throw]=throw => IndexThrow  
Threads[throw]=Thread => IndexThrow  
Threads[1]=throw => RightHandSideThrow  
Threads[1]=Thread => Okay!
```

```
Threads[throw]=throw => IndexThrow  
Threads[throw]=StringBuffer => IndexThrow  
Threads[9]=throw => RightHandSideThrow  
Threads[9]=StringBuffer => IndexOutOfBoundsException
```

```
Threads[throw]=throw => IndexThrow  
Threads[throw]=Thread => IndexThrow  
Threads[9]=throw => RightHandSideThrow  
Threads[9]=Thread => IndexOutOfBoundsException
```

The most interesting case of the lot is the one thirteenth from the end:

```
Threads[1]=StringBuffer => ArrayStoreException
```

which indicates that the attempt to store a reference to a `StringBuffer` into an array whose components are of type `Thread` throws an `ArrayStoreException`. The code is type-correct at compile time: the assignment has a left-hand side of type `Object[]` and a right-hand side of type `Object`. At run time, the first actual argument to method `testFour` is a reference to an instance of "array of `Thread`" and the third actual argument is a reference to an instance of class `StringBuffer`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.25.2 Compound Assignment Operators

All compound assignment operators require both operands to be of primitive type, except for `+=`, which allows the right-hand operand to be of any type if the left-hand operand is of type `String`.

A compound assignment expression of the form $E1 \text{ op} = E2$ is equivalent to $E1 = (T) ((E1) \text{ op } (E2))$, where T is the type of $E1$, except that $E1$ is evaluated only once. Note that the implied cast to type T may be either an identity conversion (§5.1.1) or a narrowing primitive conversion (§5.1.3). For example, the following code is correct:

```
short x = 3;  
x += 4.6;
```

and results in `x` having the value 7 because it is equivalent to:

```
short x = 3;  
x = (short)(x + 4.6);
```

At run time, the expression is evaluated in one of two ways. If the left-hand operand expression is not an array access expression, then four steps are required:

- First, the left-hand operand is evaluated to produce a variable. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason; the right-hand operand is not evaluated and no assignment occurs.
- Otherwise, the value of the left-hand operand is saved and then the right-hand operand is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and no assignment occurs.
- Otherwise, the saved value of the left-hand variable and the value of the right-hand operand are used to perform the binary operation indicated by the compound assignment operator. If this operation completes abruptly (the only possibility is an integer division by zero—see §15.16.2), then the assignment expression completes abruptly for the same reason and no assignment occurs.
- Otherwise, the result of the binary operation is converted to the type of the left-hand variable and the result of the conversion is stored into the variable.

If the left-hand operand expression is an array access expression (§15.12), then many steps are required:

- First, the array reference subexpression of the left-hand operand array access expression is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason; the index subexpression (of the left-hand operand array access expression) and the right-hand operand are not evaluated and no assignment occurs.
- Otherwise, the index subexpression of the left-hand operand array access expression is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and the right-hand operand is not evaluated and no assignment occurs.
- Otherwise, if the value of the array reference subexpression is `null`, then no assignment occurs and a `NullPointerException` is thrown.

- Otherwise, the value of the array reference subexpression indeed refers to an array. If the value of the index subexpression is less than zero, or greater than or equal to the length of the array, then no assignment occurs and an `IndexOutOfBoundsException` is thrown.
- Otherwise, the value of the index subexpression is used to select a component of the array referred to by the value of the array reference subexpression. The value of this component is saved and then the right-hand operand is evaluated. If this evaluation completes abruptly, then the assignment expression completes abruptly for the same reason and no assignment occurs. (For a simple assignment operator, the evaluation of the right-hand operand occurs before the checks of the array reference subexpression and the index subexpression, but for a compound assignment operator, the evaluation of the right-hand operand occurs after these checks.)
- Otherwise, consider the array component selected in the previous step, whose value was saved. This component is a variable; call its type *S*. Also, let *T* be the type of the left-hand operand of the assignment operator as determined at compile time.
 - If *T* is a primitive type, then *S* is necessarily the same as *T*.
 - The saved value of the array component and the value of the right-hand operand are used to perform the binary operation indicated by the compound assignment operator. If this operation completes abruptly (the only possibility is an integer division by zero—see [§15.16.2](#)), then the assignment expression completes abruptly for the same reason and no assignment occurs.
 - Otherwise, the result of the binary operation is converted to the type of the array component and the result of the conversion is stored into the array component.
 - If *T* is a reference type, then it must be `String`. Because class `String` is a `final` class, *S* must also be `String`. Therefore the run-time check that is sometimes required for the simple assignment operator is never required for a compound assignment operator.
 - The saved value of the array component and the value of the right-hand operand are used to perform the binary operation (string concatenation) indicated by the compound assignment operator (which is necessarily `+=`). If this operation completes abruptly, then the assignment expression completes abruptly for the same reason and no assignment occurs.
 - Otherwise, the `String` result of the binary operation is stored into the array component.

The rules for compound assignment to an array component are illustrated by the following example program:

```
class ArrayReferenceThrow extends RuntimeException { }

class IndexThrow extends RuntimeException { }

class RightHandSideThrow extends RuntimeException { }

class IllustrateCompoundArrayAssignment {

    static String[] strings = { "Simon", "Garfunkel" };

    static double[] doubles = { Math.E, Math.PI };
}
```



```

static String[] stringsThrow() {
    throw new ArrayReferenceThrow();
}

static double[] doublesThrow() {
    throw new ArrayReferenceThrow();
}

static int indexThrow() { throw new IndexThrow(); }

static String stringThrow() {
    throw new RightHandSideThrow();
}

static double doubleThrow() {
    throw new RightHandSideThrow();
}

static String name(Object q) {
    String sq = q.getClass().getName();
    int k = sq.lastIndexOf('.');
    return (k < 0) ? sq : sq.substring(k+1);
}

static void testEight(String[] x, double[] z, int j) {
    String sx = (x == null) ? "null" : "Strings";
    String sz = (z == null) ? "null" : "doubles";
    System.out.println();
    try {
        System.out.print(sx + "[throw]+=throw => ");
        x[indexThrow()] += stringThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sz + "[throw]+=throw => ");
        z[indexThrow()] += doubleThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }

    try {
        System.out.print(sx + "[throw]+=\"heh\" => ");
        x[indexThrow()] += "heh";
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sz + "[throw]+=12345 => ");
        z[indexThrow()] += 12345;
    }
}

```



```

        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sx + "[" + j + "]+=throw => ");
        x[j] += stringThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sz + "[" + j + "]+=throw => ");
        z[j] += doubleThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sx + "[" + j + "]+=\"heh\" => ");
        x[j] += "heh";
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print(sz + "[" + j + "]+=12345 => ");
        z[j] += 12345;
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
}

```

```

public static void main(String[] args) {
    try {
        System.out.print("throw[throw]+=throw => ");
        stringsThrow()[indexThrow()] += stringThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[throw]+=throw => ");
        doublesThrow()[indexThrow()] += doubleThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[throw]+=\"heh\" => ");
        stringsThrow()[indexThrow()] += "heh";
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
}

```

```

    try {
        System.out.print("throw[throw]+=12345 => ");
        doublesThrow()[indexThrow()] += 12345;
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]+=throw => ");
        stringsThrow()[1] += stringThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]+=throw => ");

```



```

        doublesThrow()[1] += doubleThrow();
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]+=\"heh\" => ");
        stringsThrow()[1] += "heh";
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }
    try {
        System.out.print("throw[1]+=12345 => ");
        doublesThrow()[1] += 12345;
        System.out.println("Okay!");
    } catch (Throwable e) { System.out.println(name(e)); }

    testEight(null, null, 1);
    testEight(null, null, 9);
    testEight(strings, doubles, 1);
    testEight(strings, doubles, 9);
}
}

```

This program prints:

```

throw[throw]+=throw => ArrayReferenceThrow
throw[throw]+=throw => ArrayReferenceThrow
throw[throw]+="heh" => ArrayReferenceThrow
throw[throw]+=12345 => ArrayReferenceThrow
throw[1]+=throw => ArrayReferenceThrow
throw[1]+=throw => ArrayReferenceThrow
throw[1]+="heh" => ArrayReferenceThrow
throw[1]+=12345 => ArrayReferenceThrow

```

```

null[throw]+=throw => IndexThrow
null[throw]+=throw => IndexThrow
null[throw]+="heh" => IndexThrow
null[throw]+=12345 => IndexThrow
null[1]+=throw => NullPointerException
null[1]+=throw => NullPointerException
null[1]+="heh" => NullPointerException
null[1]+=12345 => NullPointerException

```

```

null[throw]+=throw => IndexThrow
null[throw]+=throw => IndexThrow
null[throw]+="heh" => IndexThrow
null[throw]+=12345 => IndexThrow
null[9]+=throw => NullPointerException
null[9]+=throw => NullPointerException
null[9]+="heh" => NullPointerException
null[9]+=12345 => NullPointerException

```



```

Strings[throw]+=throw => IndexThrow
doubles[throw]+=throw => IndexThrow
Strings[throw]+="heh" => IndexThrow
doubles[throw]+=12345 => IndexThrow
Strings[1]+=throw => RightHandSideThrow
doubles[1]+=throw => RightHandSideThrow
Strings[1]+="heh" => Okay!
doubles[1]+=12345 => Okay!

```

```

Strings[throw]+=throw => IndexThrow
doubles[throw]+=throw => IndexThrow
Strings[throw]+="heh" => IndexThrow
doubles[throw]+=12345 => IndexThrow
Strings[9]+=throw => IndexOutOfBoundsException
doubles[9]+=throw => IndexOutOfBoundsException
Strings[9]+="heh" => IndexOutOfBoundsException
doubles[9]+=12345 => IndexOutOfBoundsException

```

The most interesting cases of the lot are tenth and eleventh from the end:

```

Strings[1]+=throw => RightHandSideThrow
doubles[1]+=throw => RightHandSideThrow

```

They are the cases where a right-hand side that throws an exception actually gets to throw the exception; moreover, they are the only such cases in the lot. This demonstrates that the evaluation of the right-hand operand indeed occurs after the checks for a null array reference value and an out-of-bounds index value.

The following program illustrates the fact that the value of the left-hand side of a compound assignment is saved before the right-hand side is evaluated:

```

class Test {
    public static void main(String[] args) {
        int k = 1;
        int[] a = { 1 };
        k += (k = 4) * (k + 2);
        a[0] += (a[0] = 4) * (a[0] + 2);
        System.out.println("k==" + k + " and a[0]==" + a[0]);
    }
}

```

This program prints:

```
k==25 and a[0]==25
```

The value 1 of `k` is saved by the compound assignment operator `+=` before its right-hand operand `(k = 4) * (k + 2)` is evaluated. Evaluation of this right-hand operand then assigns 4 to `k`, calculates the value 6 for `k + 2`, and then multiplies 4 by 6 to get 24. This is added to the saved value 1 to

get 25, which is then stored into `k` by the `+=` operator. An identical analysis applies to the case that uses `a[0]`. In short, the statements

```
k += (k = 4) * (k + 2);  
a[0] += (a[0] = 4) * (a[0] + 2);
```

behave in exactly the same manner as the statements:

```
k = k + (k = 4) * (k + 2);  
a[0] = a[0] + (a[0] = 4) * (a[0] + 2);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.26 Expression

An *Expression* is any assignment expression:

Expression:

AssignmentExpression

Unlike C and C++, the Java language has no comma operator.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


15.27 Constant Expression

ConstantExpression:

Expression

A compile-time *constant expression* is an expression denoting a value of primitive type or a `String` that is composed using only the following:

- Literals of primitive type and literals of type `String`
- Casts to primitive types and casts to type `String`
- The unary operators `+`, `-`, `~`, and `!` (but not `++` or `--`)
- The multiplicative operators `*`, `/`, and `%`
- The additive operators `+` and `-`
- The shift operators `<<`, `>>`, and `>>>`
- The relational operators `<`, `<=`, `>`, and `>=` (but not `instanceof`)
- The equality operators `==` and `!=`
- The bitwise and logical operators `&`, `^`, and `|`
- The conditional-and operator `&&` and the conditional-or operator `||`
- The ternary conditional operator `? :`
- Simple names that refer to `final` variables whose initializers are constant expressions
- Qualified names of the form *TypeName* . *Identifier* that refer to `final` variables whose initializers are constant expressions

Compile-time constant expressions are used in `case` labels in `switch` statements ([§14.9](#)) and have a special significance for assignment conversion ([§5.2](#)).

Examples of constant expressions:

```
true
(short) (1*2*3*4*5*6)
Integer.MAX_VALUE / 2
2.0 * Math.PI
"The integer " + Long.MAX_VALUE + " is mighty big."
```

... when faces of the throng turned toward him and ambiguous eyes stared into his, he assumed the most romantic of expressions ...

--F. Scott Fitzgerald, *This Side of Paradise* (1920)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Definite Assignment

All the evolution we know of proceeds from the vague to the definite.

--Charles Peirce

Each local variable must have a *definitely assigned* value when any access of its value occurs. An access to its value consists of the simple name of the variable occurring anywhere in an expression except as the left-hand operand of the simple assignment operator `=`.

A Java compiler must carry out a specific conservative flow analysis to make sure that, for every access of a local variable, the local variable is definitely assigned before the access; otherwise a compile-time error must occur.

The remainder of this chapter is devoted to a precise explanation of the words "definitely assigned before". The idea is that an assignment to the local variable must occur on every possible execution path to the access from the beginning of the constructor, method, or static initializer that contains the access. The analysis takes into account the structure of statements and expressions; it also provides a special treatment of the expression operators `!`, `&&`, `||`, and `?` `:`, the operators `&`, `|`, `^`, `==`, and `!=` with `boolean` operands, and `boolean`-valued constant expressions. For example, a Java compiler recognizes that `k` is definitely assigned before its access (as an argument of a method invocation) in the code:

```
{
    int k;
    if (v > 0 && (k = System.in.read()) >= 0)
        System.out.println(k);
}
```

because the access occurs only if the value of the expression:

```
v > 0 && (k = System.in.read()) >= 0
```

is true, and the value can be `true` only if the assignment to `k` is executed (more properly, evaluated). Similarly, a Java compiler will recognize that in the code:

```
{
    int k;
    while (true) {
        k = n;
        if (k >= 5) break;
        n = 6;
    }
    System.out.println(k);
}
```

the variable `k` is definitely assigned by the `while` statement because the condition expression `true` never has the value `false`, so only the `break` statement can cause the `while` statement to

complete normally, and `k` is definitely assigned before the `break` statement.

Except for the special treatment of certain boolean operators and of boolean-valued constant expressions, the values of expressions are not taken into account in the flow analysis. For example, a Java compiler must produce a compile-time error for the code:

```
{
    int k;
    int n = 5;
    if (n > 2)
        k = 3;
    System.out.println(k); // k is not "definitely assigned" before this
}
```

even though the value of `n` is known at compile time, and in principle it can be known at compile time that the assignment to `k` will always be executed (more properly, evaluated). A Java compiler must operate according to the rules laid out in this section. The rules recognize only constant expressions; in this example, the expression `n > 2` is not a constant expression as defined in [§15.27](#).

As another example, a Java compiler will accept the code:

```
void flow(boolean flag) {
    int k;
    if (flag)
        k = 3;
    else
        k = 4;
    System.out.println(k);
}
```

as far as definite assignment of `k` is concerned, because the rules outlined in this section allow it to tell that `k` is assigned no matter whether the flag is `true` or `false`. But the rules do not accept the variation:

```
void flow(boolean flag) {
    int k;
    if (flag)
        k = 3;
    if (!flag)
        k = 4;
    System.out.println(k); // k is not "definitely assigned" before this
}
```

and so compiling this program must cause a compile-time error to occur.

In order to precisely specify all the cases of definite assignment, the rules in this section define two technical terms:

- whether a local variable is *definitely assigned before* a statement or expression, and
- whether a local variable is *definitely assigned after* a statement or expression.

In order to specify boolean-valued expressions, the latter notion is refined into two cases:

- whether a local variable is *definitely assigned after* the expression *when true*, and
- whether a local variable is *definitely assigned after* the expression *when false*.

Here *when true* and *when false* refer to the value of the expression. For example, the local variable *k* is definitely assigned a value after evaluation of the expression

```
a && ((k=m) > 5)
```

when the expression is `true` but not when the expression is `false` (because if *a* is `false`, then the assignment to *k* is not executed (more properly, evaluated)).

The statement "*V* is definitely assigned after *X*" (where *V* is a local variable and *X* is a statement or expression) means "*V* is definitely assigned after *X* if *X* completes normally". If *X* completes abruptly, the assignment may not have occurred, and the rules stated here take this into account. A peculiar consequence of this definition is that "*V* is definitely assigned after `break;`" is always true! Because a `break` statement never completes normally, it is vacuously true that *V* has been assigned a value if the `break` statement completes normally.

To shorten the rules, the customary abbreviation "iff" is used to mean "if and only if".

Let *V* be a local variable. Let *a*, *b*, *c*, and *e* be expressions. Let *S* and *T* be statements.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1 Definite Assignment and Expressions

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.1 Boolean Constant Expressions

V is definitely assigned after any constant expression whose value is `true` when `false`. V is definitely assigned after any constant expression whose value is `false` when `true`.

A constant expression whose value is `true` never has the value `false`, and a constant expression whose value is `false` never has the value `true`, these definitions are vacuously satisfied. They are helpful in analyzing expressions involving the boolean operators `&&`, `||`, and `!` ([§16.1.3](#), [§16.1.4](#), [§16.1.5](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.2 Boolean-valued Expressions

For every boolean-valued expression:

- If the expression has no subexpressions, V is definitely assigned after the expression iff V is definitely assigned before the expression. This case applies to literals and simple names.
- Otherwise, V is definitely assigned after the expression iff V is definitely assigned after the expression when true and V is definitely assigned after the expression when false.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.3 The Boolean Operator &&

- V is definitely assigned after $a \ \&\& \ b$ when true iff V is definitely assigned after a when true or V is definitely assigned after b when true.
- V is definitely assigned after $a \ \&\& \ b$ when false iff V is definitely assigned after a when false and V is definitely assigned after b when false.
- V is definitely assigned before a iff V is definitely assigned before $a \ \&\& \ b$.
- V is definitely assigned before b iff V is definitely assigned after a when true.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.4 The Boolean Operator \parallel

- V is definitely assigned after $a \parallel b$ when true iff V is definitely assigned after a when true and V is definitely assigned after b when true.
- V is definitely assigned after $a \parallel b$ when false iff V is definitely assigned after a when false or V is definitely assigned after b when false.
- V is definitely assigned before a iff V is definitely assigned before $a \parallel b$.
- V is definitely assigned before b iff V is definitely assigned after a when false.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.5 The Boolean Operator !

- V is definitely assigned after $!a$ when true iff V is definitely assigned after a when false.
- V is definitely assigned after $!a$ when false iff V is definitely assigned after a when true.
- V is definitely assigned before a iff V is definitely assigned before $!a$.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.6 The Boolean Operator &

- V is definitely assigned after $a \ \& \ b$ when true iff V is definitely assigned after a when true or V is definitely assigned after b when true.
- V is definitely assigned after $a \ \& \ b$ when false iff at least one of the following is true:
 - V is definitely assigned after b . (Note that if V is definitely assigned after a , it follows that V is definitely assigned after b .)
 - V is definitely assigned after a when false and V is definitely assigned after b when false.
- V is definitely assigned before a iff V is definitely assigned before $a \ \& \ b$.
- V is definitely assigned before b iff V is definitely assigned after a .

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.7 The Boolean Operator \mid

- V is definitely assigned after $a \mid b$ when true iff at least one of the following is true:
 - V is definitely assigned after b . (Note that if V is definitely assigned after a , it follows that V is definitely assigned after b .)
 - V is definitely assigned after a when true and V is definitely assigned after b when true.
- V is definitely assigned after $a \mid b$ when false iff V is definitely assigned after a when false or V is definitely assigned after b when false.
- V is definitely assigned before a iff V is definitely assigned before $a \mid b$.
- V is definitely assigned before b iff V is definitely assigned after a .

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.8 The Boolean Operator \wedge

- V is definitely assigned after $a \wedge b$ when true iff at least one of the following is true:
 - V is definitely assigned after b .
 - V is definitely assigned after a when true and V is definitely assigned after b when true.
 - V is definitely assigned after a when false and V is definitely assigned after b when false.
- V is definitely assigned after $a \wedge b$ when false iff at least one of the following is true:
 - V is definitely assigned after b .
 - V is definitely assigned after a when true and V is definitely assigned after b when false.
 - V is definitely assigned after a when false and V is definitely assigned after b when true.
- V is definitely assigned before a iff V is definitely assigned before $a \wedge b$.
- V is definitely assigned before b iff V is definitely assigned after a .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.9 The Boolean Operator ==

- V is definitely assigned after $a == b$ when true iff at least one of the following is true:
 - V is definitely assigned after b .
 - V is definitely assigned after a when true and V is definitely assigned after b when false.
 - V is definitely assigned after a when false and V is definitely assigned after b when true.
- V is definitely assigned after $a == b$ when false iff at least one of the following is true:
 - V is definitely assigned after b .
 - V is definitely assigned after a when true and V is definitely assigned after b when true.
 - V is definitely assigned after a when false and V is definitely assigned after b when false.
- V is definitely assigned before a iff V is definitely assigned before $a == b$.
- V is definitely assigned before b iff V is definitely assigned after a .

{ewl msdncl.dll, ewcright, /c"Microsoft"}

16.1.10 The Boolean Operator !=

The rules for $a \neq b$ are identical to the rules for $a \wedge b$ ([§16.1.8](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.11 The Boolean Operator $?$:

Suppose that b and c are boolean-valued expressions.

- V is definitely assigned after $a ? b : c$ when true iff both of the following are true:
 - V is definitely assigned before b or V is definitely assigned after b when true.
 - V is definitely assigned before c or V is definitely assigned after c when true.
- V is definitely assigned after $a ? b : c$ when false iff both of the following are true:
 - V is definitely assigned before b or V is definitely assigned after b when false.
 - V is definitely assigned before c or V is definitely assigned after c when false.
- V is definitely assigned before a iff V is definitely assigned before $a ? b : c$.
- V is definitely assigned before b iff V is definitely assigned after a when true.
- V is definitely assigned before c iff V is definitely assigned after a when false.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.12 The Conditional Operator $?$:

Suppose that b and c are expressions that are not boolean-valued.

- V is definitely assigned after $a ? b : c$ iff both of the following are true:
 - V is definitely assigned after b .
 - V is definitely assigned after c .
- V is definitely assigned before a iff V is definitely assigned before $a ? b : c$.
- V is definitely assigned before b iff V is definitely assigned after a when true.
- V is definitely assigned before c iff V is definitely assigned after a when false.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.13 Boolean Assignment Expressions

Suppose that an assignment expression $a = b$, $a \&= b$, $a |= b$, or $a \wedge= b$ is boolean-valued.

- V is definitely assigned before a iff V is definitely assigned before the assignment expression.
- V is definitely assigned before b iff V is definitely assigned after a .
- V is definitely assigned after $a = b$ when true iff either a is V or V is definitely assigned after the right-hand operand expression when true.
- V is definitely assigned after $a = b$ when false iff either a is V or V is definitely assigned after the right-hand operand expression when false.
- V is definitely assigned after $a \&= b$ when true iff either a is V or V would be definitely assigned after $a \& b$ (in the same context) when true.
- V is definitely assigned after $a \&= b$ when false iff either a is V or V would be definitely assigned after $a \& b$ (in the same context) when false.
- V is definitely assigned after $a |= b$ when true iff either a is V or V would be definitely assigned after $a | b$ (in the same context) when true.
- V is definitely assigned after $a |= b$ when false iff either a is V or V would be definitely assigned after $a | b$ (in the same context) when false.
- V is definitely assigned after $a \wedge= b$ when true iff either a is V or V would be definitely assigned after $a \wedge b$ (in the same context) when true.
- V is definitely assigned after $a \wedge= b$ when false iff either a is V or V would be definitely assigned after $a \wedge b$ (in the same context) when false.

Note that if a is V and V is not definitely assigned before a compound assignment such as $a \&= b$, then a compile-time error will necessarily occur. The rules stated above include the disjunct " a is V " so that V will be considered to have been definitely assigned at later points in the code. Including the disjunct " a is V " does not affect the binary decision as to whether a program is acceptable or will result in a compile-time error, but it affects *how many* different points in the code may be regarded as erroneous, and so in practice it can improve the quality of error reporting.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.1.14 Other Assignment Expressions

Suppose that an assignment expression $a = b$, $a += b$, $a -= b$, $a *= b$, $a /= b$, $a \% = b$, $a < < = b$, $a > > = b$, $a > > > = b$, $a \& = b$, $a | = b$, or $a \wedge = b$ is not boolean-valued.

- V is definitely assigned after the assignment expression iff either a is V or V is definitely assigned after b .
- V is definitely assigned before a iff V is definitely assigned before the assignment expression.
- V is definitely assigned before b iff V is definitely assigned after a .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.15 Operators ++ and --

- V is definitely assigned after a preincrement, predecrement, postincrement, or postdecrement expression iff either the operand expression is V or V is definitely assigned after the operand expression.
- V is definitely assigned before the operand expression iff V is definitely assigned before the preincrement, predecrement, postincrement, or postdecrement expression.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.1.16 Other Expressions

If an expression is not boolean-valued and is not a conditional-operator expression or assignment expression, the following rules apply:

- If the expression has no subexpressions, *V* is definitely assigned after the expression iff *V* is definitely assigned before the expression. This case applies to literals, simple names, *this*, *super*, and *null*.
- If the expression has subexpressions, *V* is definitely assigned after the expression iff *V* is definitely assigned after its rightmost immediate subexpression.

For any immediate subexpression *y* of an expression *x*, *V* is definitely assigned before *y* iff *V* is definitely assigned before *x* or one of the following situations is true:

- *y* is the right-hand operand of a binary operator and *V* is definitely assigned after the left-hand operand.
- *x* is an array reference, *y* is the subexpression within the brackets, and *V* is definitely assigned after the subexpression before the brackets.
- *x* is a method invocation expression for an object; *y* is the first argument expression in the method invocation expression; there is a subexpression whose value is an object to the left of the dot, method name, and left parenthesis of the method invocation expression; and *V* is definitely assigned after this subexpression.
- *x* is a method invocation expression or class instance creation expression; *y* is an argument expression, but not the first; and *V* is definitely assigned after the argument expression to the left of *y*.
- *x* is a class instance creation expression; *y* is a dimension expression, but not the first; and *V* is definitely assigned after the dimension expression to the left of *y*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.2 Definite Assignment and Statements

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.2.1 Empty Statements

- V is definitely assigned after an empty statement iff it is definitely assigned before the empty statement.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.2 Blocks

- V is definitely assigned after an empty block iff it is definitely assigned before the empty block.
- V is definitely assigned after a nonempty block iff it is definitely assigned after the last statement in the block.
- V is definitely assigned before the first statement of the block iff it is definitely assigned before the block.
- V is definitely assigned before any other statement S of the block iff it is definitely assigned after the statement immediately preceding S in the block.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.3 Local Variable Declaration Statements

- V is definitely assigned after a local variable declaration statement that contains no initializers iff it is definitely assigned before the local variable declaration statement.
- V is definitely assigned after a local variable declaration statement that contains initializers iff either it is definitely assigned after the last initializer expression in the local variable declaration statement or the last initializer expression in the declaration is in the declarator that declares V .
- V is definitely assigned before the first initializer expression iff it is definitely assigned before the local variable declaration statement.
- V is definitely assigned before any other initializer expression e iff either it is definitely assigned after the initializer expression immediately preceding e in the local variable declaration statement or the initializer expression immediately preceding e in the local variable declaration statement is in the declarator that declares V .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.4 Labeled Statements

- V is definitely assigned after a labeled statement $L : S$ (where L is a label) iff V is definitely assigned after S and V is definitely assigned before every `break` statement that may exit the labeled statement $L : S$.
- V is definitely assigned before S iff V is definitely assigned before $L : S$.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.5 Expression Statements

- V is definitely assigned after an expression statement e ; iff it is definitely assigned after e .
- V is definitely assigned before e iff it is definitely assigned before e ; .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.6 if Statements

- V is definitely assigned after $\text{if } (e) \ S$ iff V is definitely assigned after S and V is definitely assigned after e when false.
- V is definitely assigned before e iff V is definitely assigned before $\text{if } (e) \ S$. V is definitely assigned before S iff V is definitely assigned after e when true.
- V is definitely assigned after $\text{if } (e) \ S \ \text{else } T$ iff V is definitely assigned after S and V is definitely assigned after T .
- V is definitely assigned before e iff V is definitely assigned before $\text{if } (e) \ S \ \text{else } T$. V is definitely assigned before S iff V is definitely assigned after e when true. V is definitely assigned before T iff V is definitely assigned after e when false.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.7 switch Statements

- `V` is definitely assigned after a `switch` statement iff both of the following are true:
 - Either the `switch` block is empty or `V` is definitely assigned after the last statement of the `switch` block.
 - `V` is definitely assigned before every `break` statement that may exit the `switch` statement.
- `V` is definitely assigned before the switch expression iff `V` is definitely assigned before the `switch` statement.
- `V` is definitely assigned before a statement or local variable declaration statement `S` in the switch block iff at least one of the following is true:
 - `V` is definitely assigned after the switch expression.
 - `S` is not labeled by a `case` or `default` label and `V` is definitely assigned after the preceding statement.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.2.8 while Statements

- V is definitely assigned after `while (e) S` iff V is definitely assigned after e when false and V is definitely assigned before every `break` statement that may exit the `while` statement.
- V is definitely assigned before e iff V is definitely assigned before the `while` statement.
- V is definitely assigned before S iff V is definitely assigned after e when true.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.2.9 do Statements

- V is definitely assigned after `do S while (e) ;` iff V is definitely assigned after e when false and V is definitely assigned before every `break` statement that may exit the `do` statement.
- V is definitely assigned before S iff V is definitely assigned before the `do` statement.
- V is definitely assigned before e iff V is definitely assigned after S and V is definitely assigned before every `continue` statement that may exit the body of the `do` statement.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.10 for Statements

- V is definitely assigned after a `for` statement iff both of the following are true:
 - Either a condition expression is not present or V is definitely assigned after the condition expression when false.
 - V is definitely assigned before every `break` statement that may exit the `for` statement.
- V is definitely assigned before the initialization part of the `for` statement iff V is definitely assigned before the `for` statement.
- V is definitely assigned before the condition part of the `for` statement iff V is definitely assigned after the initialization part of the `for` statement.
- V is definitely assigned before the contained statement iff either of the following is true:
 - A condition expression is present and V is definitely assigned after the condition expression when true.
 - No condition expression is present and V is definitely assigned after the initialization part of the `for` statement.
- V is definitely assigned before the incrementation part of the `for` statement iff V is definitely assigned after the contained statement and V is definitely assigned before every `continue` statement that may exit the body of the `for` statement.

16.2.10.1 Initialization Part

- If the initialization part of the `for` statement is a local variable declaration statement, the rules of [§16.2.3](#) apply.
- Otherwise, if the initialization part is empty, then V is definitely assigned after the initialization part iff V is definitely assigned before the initialization part.
- Otherwise, three rules apply:
 - V is definitely assigned after the initialization part iff V is definitely assigned after the last expression statement in the initialization part.
 - V is definitely assigned before the first expression statement in the initialization part iff V is definitely assigned before the initialization part.
 - V is definitely assigned before an expression statement E other than the first in the initialization part iff V is definitely assigned after the expression statement immediately preceding E .

16.2.10.2 Incrementation Part

- If the incrementation part of the `for` statement is empty, then V is definitely assigned after the incrementation part iff V is definitely assigned before the incrementation part.
- Otherwise, three rules apply:
 - V is definitely assigned after the incrementation part iff V is definitely assigned after the last expression statement in the incrementation part.
 - V is definitely assigned before the first expression statement in the incrementation part iff V is definitely assigned before the incrementation part.
 - V is definitely assigned before an expression statement E other than the first in the incrementation part iff V is definitely assigned after the expression statement immediately preceding E .

preceding *E*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.11 break, continue, return, and throw Statements

- By convention, we say that *V* is definitely assigned after any `break`, `continue`, `return`, or `throw` statement. The notion that a variable is "definitely assigned after" a statement or expression really means "is definitely assigned after the statement or expression completes normally". Because a `break`, `continue`, `return`, or `throw` statement never completes normally, it vacuously satisfies this notion.
- In a `return` statement with an expression or a `throw` statement, *V* is definitely assigned before the expression iff *V* is definitely assigned before the `return` or `throw` statement.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

16.2.12 synchronized Statements

- V is definitely assigned after `synchronized (e) S` iff V is definitely assigned after S .
- V is definitely assigned before e iff V is definitely assigned before the statement `synchronized (e) S`.
- V is definitely assigned before S iff V is definitely assigned after e .

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


16.2.13 try Statements

- V is definitely assigned after a `try` statement iff one of the following is true:
 - V is definitely assigned after the `try` block and V is definitely assigned after every `catch` block in the `try` statement.
 - The `try` statement has a `finally` block and V is definitely assigned after the `finally` block.
- V is definitely assigned before the `try` block iff V is definitely assigned before the `try` statement.
- V is definitely assigned before a `catch` block iff V is definitely assigned before the `try` statement.

V is definitely assigned before a `finally` block iff V is definitely assigned before the `try` statement.

I resolved to assign Bartleby a corner by the folding doors . . .
--Herman Melville, *Bartleby, the Scrivener* (1853)

It does not strike me that there is anything definite about that.
--Herman Melville, *Bartleby, the Scrivener* (1853)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Threads and Locks

*And oft-times in the most forbidding den
Of solitude, with love of science strong,
How patiently the yoke of thought they bear;
How subtly glide its finest threads along!*

--William Wordsworth, Monks and Schoolmen, in Ecclesiastical Sonnets (1822)

While most of the discussion in the preceding chapters is concerned only with the behavior of Java code as executed a single statement or expression at a time, that is, by a single *thread*, each Java Virtual Machine can support many threads of execution at once. These threads independently execute Java code that operates on Java values and objects residing in a shared main memory. Threads may be supported by having many hardware processors, by time-slicing a single hardware processor, or by time-slicing many hardware processors.

Java supports the coding of programs that, though concurrent, still exhibit deterministic behavior, by providing mechanisms for *synchronizing* the concurrent activity of threads. To synchronize threads, Java uses *monitors*, which are a high-level mechanism for allowing only one thread at a time to execute a region of code protected by the monitor. The behavior of monitors is explained in terms of *locks*; there is a lock associated with each object.

The `synchronized` statement (§14.17) performs two special actions relevant only to multithreaded operation: (1) after computing a reference to an object but before executing its body, it *locks* a lock associated with the object, and (2) after execution of the body has completed, either normally or abruptly, it *unlocks* that same lock. As a convenience, a method may be declared `synchronized`; such a method behaves as if its body were contained in a `synchronized` statement.

The methods `wait` (§20.1.6, §20.1.7, §20.1.8), `notify` (§20.1.9), and `notifyAll` (§20.1.10) of class `Object` support an efficient transfer of control from one thread to another. Rather than simply "spinning" (repeatedly locking and unlocking an object to see whether some internal state has changed), which consumes computational effort, a thread can suspend itself using `wait` until such time as another thread awakens it using `notify`. This is especially appropriate in situations where threads have a producer-consumer relationship (actively cooperating on a common goal) rather than a mutual exclusion relationship (trying to avoid conflicts while sharing a common resource).

As a thread executes code, it carries out a sequence of actions. A thread may *use* the value of a variable or *assign* it a new value. (Other actions include arithmetic operations, conditional tests, and method invocations, but these do not involve variables directly.) If two or more concurrent threads act on a shared variable, there is a possibility that the actions on the variable will produce timing-dependent results. This dependence on timing is inherent in concurrent programming, producing one of the few places in Java where the result of executing a program is not determined solely by this specification.

Each thread has a working memory, in which it may keep copies of the values of variables from the main memory that is shared between all threads. To access a shared variable, a thread usually first obtains a lock and flushes its working memory. This guarantees that shared values will be thereafter be loaded from the shared main memory to the thread's working memory. When a thread unlocks a lock it guarantees the values it holds in its working memory will be written back to the main memory.

This chapter explains the interaction of threads with the main memory, and thus with each other, in terms of certain low-level actions. There are rules about the order in which these actions may occur. These rules impose constraints on any implementation of Java, and a Java programmer may rely on the rules to predict the possible behaviors of a concurrent Java program. The rules do, however, intentionally give the implementor certain freedoms; the intent is to permit certain standard hardware

and software techniques that can greatly improve the speed and efficiency of concurrent code.

Briefly put, these are the important consequences of the rules:

- Proper use of synchronization constructs will allow reliable transmission of values or sets of values from one thread to another through shared variables.
- When a thread uses the value of a variable, the value it obtains is in fact a value stored into the variable by that thread or by some other thread. This is true even if the program does not contain code for proper synchronization. For example, if two threads store references to different objects into the same reference value, the variable will subsequently contain a reference to one object or the other, not a reference to some other object or a corrupted reference value. (There is a special exception for `long` and `double` values; see [§17.4](#).)
- In the absence of explicit synchronization, a Java implementation is free to update the main memory in an order that may be surprising. Therefore the programmer who prefers to avoid surprises should use explicit synchronization.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.1 Terminology and Framework

A *variable* is any location within a Java program that may be stored into. This includes not only class variables and instance variables but also components of arrays. Variables are kept in a *main memory* that is shared by all threads. Because it is impossible for one thread to access parameters or local variables of another thread, it doesn't matter whether parameters and local variables are thought of as residing in the shared main memory or in the working memory of the thread that owns them.

Every thread has a *working memory* in which it keeps its own *working copy* of variables that it must use or assign. As the thread executes a Java program, it operates on these working copies. The main memory contains the *master copy* of every variable. There are rules about when a thread is permitted or required to transfer the contents of its working copy of a variable into the master copy or vice versa.

The main memory also contains *locks*; there is one lock associated with each object. Threads may compete to acquire a lock.

For the purposes of this chapter, the verbs *use*, *assign*, *load*, *store*, *lock*, and *unlock* name *actions* that a thread can perform. The verbs *read*, *write*, *lock*, and *unlock* name actions that the main memory subsystem can perform. Each of these actions is *atomic* (indivisible).

A *use* or *assign* action is a tightly coupled interaction between a thread's execution engine and the thread's working memory. A *lock* or *unlock* action is a tightly coupled interaction between a thread's execution engine and the main memory. But the transfer of data between the main memory and a thread's working memory is loosely coupled. When data is copied from the main memory to a working memory, two actions must occur: a *read* action performed by the main memory followed some time later by a corresponding *load* action performed by the working memory. When data is copied from a working memory to the main memory, two actions must occur: a *store* action performed by the working memory followed some time later by a corresponding *write* action performed by the main memory. There may be some transit time between main memory and a working memory, and the transit time may be different for each transaction; thus actions initiated by a thread on different variables may viewed by another thread as occurring in a different order. For each variable, however, the actions in main memory on behalf of any one thread are performed in the same order as the corresponding actions by that thread. (This is explained in greater detail below.)

A single Java thread issues a stream of *use*, *assign*, *lock*, and *unlock* actions as dictated by the semantics of the Java program it is executing. The underlying Java implementation is then required additionally to perform appropriate *load*, *store*, *read*, and *write* actions so as to obey a certain set of constraints, explained below. If the Java implementation correctly follows these rules and the Java application programmer follows certain other rules of programming, then data can be reliably transferred between threads through shared variables. The rules are designed to be "tight" enough to make this possible but "loose" enough to allow hardware and software designers considerable freedom to improve speed and throughput through such mechanisms as registers, queues, and caches.

Here are the detailed definitions of each of the actions:

- A *use* action (by a thread) transfers the contents of the thread's working copy of a variable to the thread's execution engine. This action is performed whenever a thread executes a virtual machine instruction that uses the value of a variable.
- An *assign* action (by a thread) transfers a value from the thread's execution engine into the thread's working copy of a variable. This action is performed whenever a thread executes a virtual machine instruction that assigns to a variable.
- A *read* action (by the main memory) transmits the contents of the master copy of a variable to a thread's working memory for use by a later *load* action.
- A *load* action (by a thread) puts a value transmitted from main memory by a *read* action into the thread's working copy of a variable.

- A *store* action (by a thread) transmits the contents of the thread's working copy of a variable to main memory for use by a later *write* action.
- A *write* action (by the main memory) puts a value transmitted from the thread's working memory by a *store* action into the master copy of a variable in main memory.
- A *lock* action (by a thread tightly synchronized with main memory) causes a thread to acquire one claim on a particular lock.
- An *unlock* action (by a thread tightly synchronized with main memory) causes a thread to release one claim on a particular lock.

Thus the interaction of a thread with a variable over time consists of a sequence of *use*, *assign*, *load*, and *store* actions. Main memory performs a *read* action for every *load* and a *write* action for every *store*. A thread's interactions with a lock over time consists of a sequence of *lock* and *unlock* actions. All the globally visible behavior of a thread thus comprises all the thread's actions on variables and locks.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.2 Execution Order

The rules of execution order constrain the order in which certain events may occur. There are four general constraints on the relationships among actions:

- The actions performed by any one thread are totally ordered; that is, for any two actions performed by a thread, one action precedes the other.
- The actions performed by the main memory for any one variable are totally ordered; that is, for any two actions performed by the main memory on the same variable, one action precedes the other.
- The actions performed by the main memory for any one lock are totally ordered; that is, for any two actions performed by the main memory on the same lock, one action precedes the other.
- It is not permitted for an action to follow itself.

The last rule may seem trivial, but it does need to be stated separately and explicitly for completeness. Without it, it would be possible to propose a set of actions by two or more threads and precedence relationships among the actions that would satisfy all the other rules but would require an action to follow itself.

Threads do not interact directly; they communicate only through the shared main memory. The relationships between the actions of a thread and the actions of main memory are constrained in three ways:

- Each *lock* or *unlock* action is performed jointly by some thread and the main memory.
- Each *load* action by a thread is uniquely paired with a *read* action by the main memory such that the *load* action follows the *read* action.
- Each *store* action by a thread is uniquely paired with a *write* action by the main memory such that the *write* action follows the *store* action.

Most of the rules in the following sections further constrain the order in which certain actions take place. A rule may state that one action must precede or follow some other action. Note that this relationship is transitive: if action *A* must precede action *B*, and *B* must precede *C*, then *A* must precede *C*. The programmer must remember that these rules are the *only* constraints on the ordering of actions; if no rule or combination of rules implies that action *A* must precede action *B*, then a Java implementation is free to perform action *B* before action *A*, or to perform action *B* concurrently with action *A*. This freedom can be the key to good performance. Conversely, an implementation is not required to take advantage of all the freedoms given it.

In the rules that follow, the phrasing "*B* must intervene between *A* and *C*" means that action *B* must follow action *A* and precede action *C*.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.3 Rules about Variables

Let T be a thread and V be a variable. There are certain constraints on the actions performed by T with respect to V :

- An *use* or *assign* by T of V is permitted only when dictated by execution by T of the Java program according to the standard Java execution model. For example, an occurrence of V as an operand of the $+$ operator requires that a single *use* action occur on V ; an occurrence of V as the left-hand operand of the assignment operator $=$ requires that a single *assign* action occur. All *use* and *assign* actions by a given thread must occur in the order specified by the program being executed by the thread. If the following rules forbid T to perform a required *use* as its next action, it may be necessary for T to perform a *load* first in order to make progress.
- A *store* action by T on V must intervene between an *assign* by T of V and a subsequent *load* by T of V . (Less formally: a thread is not permitted to lose its most recent *assign*.)
- An *assign* action by T on V must intervene between a *load* or *store* by T of V and a subsequent *store* by T of V . (Less formally: a thread is not permitted to write data from its working memory back to main memory for no reason.)
- After a thread is created, it must perform an *assign* or *load* action on a variable before performing a *use* or *store* action on that variable. (Less formally: a new thread starts with an empty working memory.)
- After a variable is created, every thread must perform an *assign* or *load* action on that variable before performing a *use* or *store* action on that variable. (Less formally: a new variable is created only in main memory and is not initially in any thread's working memory.)

Provided that all the constraints above and below are obeyed, a *load* or *store* action may be issued at any time by any thread on any variable, at the whim of the implementation.

There are also certain constraints on the *read* and *write* actions performed by main memory:

- For every *load* action performed by any thread T on its working copy of a variable V , there must be a corresponding preceding *read* action by the main memory on the master copy of V , and the *load* action must put into the working copy the data transmitted by the corresponding *read* action.
- For every *store* action performed by any thread T on its working copy of a variable V , there must be a corresponding following *write* action by the main memory on the master copy of V , and the *write* action must put into the master copy the data transmitted by the corresponding *store* action.
- Let action A be a *load* or *store* by thread T on variable V , and let action P be the corresponding *read* or *write* by the main memory on variable V . Similarly, let action B be some other *load* or *store* by thread T on that same variable V , and let action Q be the corresponding *read* or *write* by the main memory on variable V . If A precedes B , then P must precede Q . (Less formally: actions on the master copy of any given variable on behalf of a thread are performed by the main memory in exactly the order that the thread requested.)

Note that this last rule applies *only* to actions by a thread on the *same* variable. However, there is a more stringent rule for `volatile` variables ([§17.7](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.4 Nonatomic Treatment of `double` and `long` Variables

If a `double` or `long` variable is not declared `volatile`, then for the purposes of *load*, *store*, *read*, and *write* actions they are treated as if they were two variables of 32 bits each: wherever the rules require one of these actions, two such actions are performed, one for each 32-bit half. The manner in which the 64 bits of a `double` or `long` variable are encoded into two 32-bit quantities is implementation-dependent.

This matters only because a *read* or *write* of a `double` or `long` variable may be handled by an actual main memory as two 32-bit *read* or *write* actions that may be separated in time, with other actions coming between them. Consequently, if two threads concurrently assign distinct values to the same shared non-`volatile` `double` or `long` variable, a subsequent use of that variable may obtain a value that is not equal to either of the assigned values, but some implementation-dependent mixture of the two values.

An implementation is free to implement *load*, *store*, *read*, and *write* actions for `double` and `long` values as atomic 64-bit actions; in fact, this is strongly encouraged. The model divides them into 32-bit halves for the sake of several currently popular microprocessors that fail to provide efficient atomic memory transactions on 64-bit quantities. It would have been simpler for Java to define all memory transactions on single variables as atomic; this more complex definition is a pragmatic concession to current hardware practice. In the future this concession may be eliminated. Meanwhile, programmers are cautioned always to explicitly synchronize access to shared `double` and `long` variables.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.5 Rules about Locks

*By the pricking of my thumbs,
Something wicked this way comes.
Open, locks,
Whoever knocks!*

--William Shakespeare, Macbeth, Act IV, scene i

Let T be a thread and L be a lock. There are certain constraints on the actions performed by T with respect to L :

- A *lock* action by T on L may occur only if, for every thread S other than T , the number of preceding *unlock* actions by S on L equals the number of preceding *lock* actions by S on L . (Less formally: only one thread at a time is permitted to lay claim to a lock, and moreover a thread may acquire the same lock multiple times and doesn't relinquish ownership of it until a matching number of *unlock* actions have been performed.)
- An *unlock* action by thread T on lock L may occur only if the number of preceding *unlock* actions by T on L is strictly less than the number of preceding *lock* actions by T on L . (Less formally: a thread is not permitted to unlock a lock it doesn't own.)

With respect to a lock, the *lock* and *unlock* actions performed by all the threads are performed in some total sequential order. This total order must be consistent with the total order on the actions of each thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

17.6 Rules about the Interaction of Locks and Variables

Let T be any thread, let V be any variable, and let L be any lock. There are certain constraints on the actions performed by T with respect to V and L :

- Between an *assign* action by T on V and a subsequent *unlock* action by T on L , a *store* action by T on V must intervene; moreover, the *write* action corresponding to that *store* must precede the *unlock* action, as seen by main memory. (Less formally: if a thread is to perform an *unlock* action on *any* lock, it must first copy *all* assigned values in its working memory back out to main memory.)
- Between a *lock* action by T on L and a subsequent *use* or *store* action by T on a variable V , an *assign* or *load* action on V must intervene; moreover, if it is a *load* action, then the *read* action corresponding to that *load* must follow the *lock* action, as seen by main memory. (Less formally: a *lock* action acts as if it flushes *all* variables from the thread's working memory; before use they must be assigned or loaded from main memory .)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

17.7 Rules for Volatile Variables

If a variable is declared `volatile`, then additional constraints apply to the actions of each thread. Let T be a thread and let V and W be volatile variables.

- An *use* action by T on V is permitted only if the previous action by T on V was *load*, and a *load* action by T on V is permitted only if the next action by T on V is *use*. The *use* action is said to be "associated" with the *read* action that corresponds to the *load*.
- A *store* action by T on V is permitted only if the previous action by T on V was *assign*, and an *assign* action by T on V is permitted only if the next action by T on V is *store*. The *assign* action is said to be "associated" with the *write* action that corresponds to the *store*.
- Let action A be a *use* or *assign* by thread T on variable V , let action F be the *load* or *store* associated with A , and let action P be the *read* or *write* of V that corresponds to F . Similarly, let action B be a *use* or *assign* by thread T on variable W , let action G be the *load* or *store* associated with B , and let action Q be the *read* or *write* of W that corresponds to G . If A precedes B , then P must precede Q . (Less formally: actions on the master copies of volatile variables on behalf of a thread are performed by the main memory in exactly the order that the thread requested.)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

17.8 Prescient Store Actions

If a variable is not declared `volatile`, then the rules in the previous sections are relaxed slightly to allow *store* actions to occur earlier than would otherwise be permitted. The purpose of this relaxation is to allow optimizing Java compilers to perform certain kinds of code rearrangement that preserve the semantics of properly synchronized programs but might be caught in the act of performing memory actions out of order by programs that are not properly synchronized.

Suppose that a *store* by *T* of *V* would follow a particular *assign* by *T* of *V* according to the rules of the previous sections, with no intervening *load* or *assign* by *T* of *V*. Then that *store* action would send to the main memory the value that the *assign* action put into the working memory of thread *T*. The special rule allows the *store* action to instead occur before the *assign* action, if the following restrictions are obeyed:

- If the *store* action occurs, the *assign* is bound to occur. (Remember, these are restrictions on what actually happens, not on what a thread plans to do. No fair performing a *store* and then throwing an exception before the *assign* occurs!)
- No *lock* action intervenes between the relocated *store* and the *assign*.
- No *load* of *V* intervenes between the relocated *store* and the *assign*.
- No other *store* of *V* intervenes between the relocated *store* and the *assign*.
- The *store* action sends to the main memory the value that the *assign* action will put into the working memory of thread *T*.

This last property inspires us to call such an early *store* action *prescient*: it has to know ahead of time, somehow, what value will be stored by the *assign* that it should have followed. In practice, optimized compiled code will compute such values early (which is permitted if, for example, the computation has no side effects and throws no exceptions), store them early (before entering a loop, for example), and keep them in working registers for later use within the loop.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.9 Discussion

Any association between locks and variables is purely conventional. Locking any lock conceptually flushes *all* variables from a thread's working memory, and unlocking any lock forces the writing out to main memory of *all* variables that the thread has assigned. That a lock may be associated with a particular object or a class is purely a convention. In some applications, it may be appropriate always to lock an object before accessing any of its instance variables, for example; synchronized methods are a convenient way to follow this convention. In other applications, it may suffice to use a single lock to synchronize access to a large collection of objects.

If a thread uses a particular shared variable only after locking a particular lock and before the corresponding unlocking of that same lock, then the thread will read the shared value of that variable from main memory after the *lock* action, if necessary, and will copy back to main memory the value most recently assigned to that variable before the *unlock* action. This, in conjunction with the mutual exclusion rules for locks, suffices to guarantee that values are correctly transmitted from one thread to another through shared variables.

The rules for `volatile` variables effectively require that main memory be touched exactly once for each *use* or *assign* of a `volatile` variable by a thread, and that main memory be touched in exactly the order dictated by the thread execution semantics. However, such memory actions are not ordered with respect to *read* and *write* actions on nonvolatile variables.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.10 Example: Possible Swap

Consider a class that has class variables `a` and `b` and methods `hither` and `yon`:

```
class Sample {
    int a = 1, b = 2;
    void hither() {
        a = b;
    }
    void yon() {
        b = a;
    }
}
```

Now suppose that two threads are created, and that one thread calls `hither` while the other thread calls `yon`. What is the required set of actions and what are the ordering constraints?

Let us consider the thread that calls `hither`. According to the rules, this thread must perform an *use* of `b` followed by an *assign* of `a`. That is the bare minimum required to execute a call to the method `hither`.

Now, the first action on variable `b` by the thread cannot be *use*. But it may be *assign* or *load*. An *assign* to `b` cannot occur because the program text does not call for such an *assign* action, so a *load* of `b` is required. This *load* action by the thread in turn requires a preceding *read* action for `b` by the main memory.

The thread may optionally *store* the value of `a` after the *assign* has occurred. If it does, then the *store* action in turn requires a following *write* action for `a` by the main memory.

The situation for the thread that calls `yon` is similar, but with the roles of `a` and `b` exchanged.

The total set of actions may be pictured as follows:

{`ewc msdn cd, EWGraphic, LNG10u 0 /a "langref7ANC.BMP"`}

Here an arrow from action *A* to action *B* indicates that *A* must precede *B*.

In what order may the actions by the main memory occur? The only constraint is that it is not possible both for the *write* of `a` to precede the *read* of `a` and for the *write* of `b` to precede the *read* of `b`, because the causality arrows in the diagram would form a loop so that an action would have to precede itself, which is not allowed. Assuming that the optional *store* and *write* actions are to occur, there are three possible orderings in which the main memory might legitimately perform its actions. Let `ha` and `hb` be the working copies of `a` and `b` for the `hither` thread, let `ya` and `yb` be the working copies for the `yon` thread, and let `ma` and `mb` be the master copies in main memory. Initially `ma=1` and `mb=2`. Then the three possible orderings of actions and the resulting states are as follows:

- `write a`{`ewc msdn cd, EWGraphic, LNG10u 1 /a "langref.BMP"`}`read a`, `read b`{`ewc msdn cd, EWGraphic, LNG10u 2 /a "langref.BMP"`}`write b` (then `ha=2`, `hb=2`, `ma=2`, `mb=2`, `ya=2`, `yb=2`)
- `read a`{`ewc msdn cd, EWGraphic, LNG10u 3 /a "langref.BMP"`}`write a`, `write b`{`ewc msdn cd, EWGraphic, LNG10u 4 /a "langref.BMP"`}`read b` (then `ha=1`, `hb=1`, `ma=1`, `mb=1`, `ya=1`, `yb=1`)
- `read a`{`ewc msdn cd, EWGraphic, LNG10u 5 /a "langref.BMP"`}`write a`, `read b`{`ewc`

msdncd, EWGraphic, LNG10u 6 /a "langref.BMP"}write b (then ha=2, hb=2, ma=2, mb=1, ya=1, yb=1)

Thus the net result might be that, in main memory, *b* is copied into *a*, *a* is copied into *b*, or the values of *a* and *b* are swapped; moreover, the working copies of the variables might or might not agree. It would be incorrect, of course, to assume that any one of these outcomes is more likely than another. This is one place in which the behavior of a Java program is necessarily timing-dependent.

Of course, an implementation might also choose not to perform the *store* and *write* actions, or only one of the two pairs, leading to yet other possible results.

Now suppose that we modify the example to use *synchronized* methods:

```
class SynchSample {
    int a = 1, b = 2;
    synchronized void hither() {
        a = b;
    }
    synchronized void yon() {
        b = a;
    }
}
```

Let us again consider the thread that calls *hither*. According to the rules, this thread must perform a *lock* action (on the class object for class *SynchSample*) before the body of method *hither* is executed. This is followed by a *use* of *b* and then an *assign* of *a*. Finally, an *unlock* action on the class object must be performed after the body of method *hither* completes. That is the bare minimum required to execute a call to the method *hither*.

As before, a *load* of *b* is required, which in turn requires a preceding *read* action for *b* by the main memory. Because the *load* follows the *lock* action, the corresponding *read* must also follow the *lock* action.

Because an *unlock* action follows the *assign* of *a*, a *store* action on *a* is mandatory, which in turn requires a following *write* action for *a* by the main memory. The *write* must precede the *unlock* action.

The situation for the thread that calls *yon* is similar, but with the roles of *a* and *b* exchanged.

The total set of actions may be pictured as follows:

{ewc msdncd, EWGraphic, LNG10u 7 /a "langref7ANC1.BMP"}

The *lock* and *unlock* actions provide further constraints on the order of actions by the main memory; the *lock* action by one thread cannot occur between the *lock* and *unlock* actions of the other thread. Moreover, the *unlock* actions require that the *store* and *write* actions occur. It follows that only two sequences are possible:

- *write* a{ewc msdncd, EWGraphic, LNG10u 8 /a "langref.BMP"}*read* a, *read* b{ewc msdncd, EWGraphic, LNG10u 9 /a "langref.BMP"}*write* b (then ha=2, hb=2, ma=2, mb=2, ya=2, yb=2)
- *read* a{ewc msdncd, EWGraphic, LNG10u 10 /a "langref.BMP"}*write* a, *write* b{ewc msdncd, EWGraphic, LNG10u 11 /a "langref.BMP"}*read* b (then ha=1, hb=1, ma=1, mb=1, ya=1, yb=1)

While the resulting state is timing-dependent, it can be seen that the two threads will necessarily

agree on the values of a and b .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

17.11 Example: Out-of-Order Writes

This example is similar to that in the preceding section, except that one method assigns to both variables and the other method reads both variables. Consider a class that has class variables `a` and `b` and methods `to` and `fro`:

```
class Simple {
    int a = 1, b = 2;
    void to() {
        a = 3;
        b = 4;
    }
    void fro() {
        System.out.println("a= " + a + ", b=" + b);
    }
}
```

Now suppose that two threads are created, and that one thread calls `to` while the other thread calls `fro`. What is the required set of actions and what are the ordering constraints?

Let us consider the thread that calls `to`. According to the rules, this thread must perform an *assign* of `a` followed by an *assign* of `b`. That is the bare minimum required to execute a call to the method `to`. Because there is no synchronization, it is at the option of the implementation whether or not to *store* the assigned values back to main memory! Therefore the thread that calls `fro` may obtain either 1 or 3 for the value of `a`, and independently may obtain either 2 or 4 for the value of `b`.

Now suppose that `to` is *synchronized* but `fro` is not:

```
class SynchSimple {
    int a = 1, b = 2;
    synchronized void to() {
        a = 3;
        b = 4;
    }
    void fro() {
        System.out.println("a= " + a + ", b=" + b);
    }
}
```

In this case the method `to` will be forced to *store* the assigned values back to main memory before the *unlock* action at the end of the method. The method `fro` must, of course, use `a` and `b` (in that order) and so must *load* values for `a` and `b` from main memory.

The total set of actions may be pictured as follows:

{ewc msdncd, EWGraphic, LNG11u 0 /a "langref7ANC2.BMP"}

Here an arrow from action *A* to action *B* indicates that *A* must precede *B*.

In what order may the actions by the main memory occur? Note that the rules do not require that *write* `a` occur before *write* `b`; neither do they require that *read* `a` occur before *read* `b`. Also, even

though method `to` is `synchronized`, method `fro` is not `synchronized`, so there is nothing to prevent the *read* actions from occurring between the *lock* and *unlock* actions. (The point is that declaring one method `synchronized` does not of itself make that method behave as if it were atomic.)

As a result, the method `fro` could still obtain either 1 or 3 for the value of `a`, and independently could obtain either 2 or 4 for the value of `b`. In particular, `fro` might observe the value 1 for `a` and 4 for `b`. Thus, even though `to` does an *assign* to `a` and then an *assign* to `b`, the *write* actions to main memory may be observed by another thread to occur as if in the opposite order.

Finally, suppose that `to` and `fro` are both `synchronized`:

```
class SynchSynchSimple {
    int a = 1, b = 2;
    synchronized void to() {
        a = 3;
        b = 4;
    }
    synchronized void fro() {
        System.out.println("a= " + a + ", b=" + b);
    }
}
```

In this case, the actions of method `fro` cannot be interleaved with the actions of method `to`, and so `fro` will print either "a=1, b=2" or "a=3, b=4".

{ewl msdncd.dll, ewcright, /c"Microsoft"}

17.12 Threads

*They plant dead trees for living, and the dead
They string together with a living thread . . .
But in no hush they string it . . . With a laugh, . . .
They bring the telephone and telegraph.
--Robert Frost, *The Line-gang* (1920)*

Threads are created and managed by the built-in classes `Thread` ([§20.20](#)) and `ThreadGroup` ([§20.21](#)). Creating a `Thread` object creates a thread and that is the only way to create a thread. When the thread is created, it is not yet active; it begins to run when its `start` method ([§20.20.14](#)) is called.

Every thread has a *priority*. When there is competition for processing resources, threads with higher priority are generally executed in preference to threads with lower priority. Such preference is not, however, a guarantee that the highest priority thread will always be running, and thread priorities cannot be used to reliably implement mutual exclusion.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.13 Locks and Synchronization

There is a lock associated with every object. The Java language does not provide a way to perform separate *lock* and *unlock* actions; instead, they are implicitly performed by high-level constructs that arrange always to pair such actions correctly. (The Java Virtual Machine, however, provides separate *monitorenter* and *monitorexit* instructions that implement the *lock* and *unlock* actions.)

The `synchronized` statement (§14.17) computes a reference to an object; it then attempts to perform a *lock* action on that object and does not proceed further until the *lock* action has successfully completed. (A *lock* action may be delayed because the rules about locks can prevent the main memory from participating until some other thread is ready to perform one or more *unlock* actions.) After the lock action has been performed, the body of the `synchronized` statement is executed. If execution of the body is ever completed, either normally or abruptly, an *unlock* action is automatically performed on that same lock.

A `synchronized` method (§8.4.3.5) automatically performs a *lock* action when it is invoked; its body is not executed until the *lock* action has successfully completed. If the method is an instance method, it locks the lock associated with the instance for which it was invoked (that is, the object that will be known as `this` during execution of the body of the method). If the method is `static`, it locks the lock associated with the `Class` object that represents the class in which the method is defined. If execution of the method's body is ever completed, either normally or abruptly, an *unlock* action is automatically performed on that same lock.

Best practice is that if a variable is ever to be assigned by one thread and used or assigned by another, then all accesses to that variable should be enclosed in `synchronized` methods or `synchronized` statements.

Java does not prevent, nor require detection of, deadlock conditions. Programs where threads hold (directly or indirectly) locks on multiple objects should use conventional techniques for deadlock avoidance, creating higher-level locking primitives that don't deadlock, if necessary.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


17.14 Wait Sets and Notification

Every object, in addition to having an associated lock, has an associated *wait set*, which is a set of threads. When an object is first created, its wait set is empty.

Wait sets are used by the methods `wait` (§20.1.6, §20.1.7, §20.1.8), `notify` (§20.1.9), and `notifyAll` (§20.1.10) of class `Object`. These methods also interact with the scheduling mechanism for threads (§20.20).

The method `wait` should be called for an object only when the current thread (call it *T*) has already locked the object's lock. Suppose that thread *T* has in fact performed *N* *lock* actions that have not been matched by *unlock* actions. The `wait` method then adds the current thread to the wait set for the object, disables the current thread for thread scheduling purposes, and performs *N* *unlock* actions to relinquish the lock. The thread *T* then lies dormant until one of three things happens:

- Some other thread invokes the `notify` method for that object and thread *T* happens to be the one arbitrarily chosen as the one to notify.
- Some other thread invokes the `notifyAll` method for that object.
- If the call by thread *T* to the `wait` method specified a timeout interval, the specified amount of real time has elapsed.

The thread *T* is then removed from the wait set and re-enabled for thread scheduling. It then locks the object again (which may involve competing in the usual manner with other threads); once it has gained control of the lock, it performs {ewc msdn cd, EWGraphic, LNG14u 0 /a "langref7ANC3.BMP"} additional *lock* actions and then returns from the invocation of the `wait` method. Thus, on return from the `wait` method, the state of the object's lock is exactly as it was when the `wait` method was invoked.

The `notify` method should be called for an object only when the current thread has already locked the object's lock. If the wait set for the object is not empty, then some arbitrarily chosen thread is removed from the wait set and re-enabled for thread scheduling. (Of course, that thread will not be able to proceed until the current thread relinquishes the object's lock.)

The `notifyAll` method should be called for an object only when the current thread has already locked the object's lock. Every thread in the wait set for the object is removed from the wait set and re-enabled for thread scheduling. (Of course, those threads will not be able to proceed until the current thread relinquishes the object's lock.)

*These pearls of thought in Persian gulfs were bred,
Each softly lucent as a rounded moon;
The diver Omar plucked them from their bed,
Fitzgerald strung them on an English thread.*
--James Russell Lowell, in a copy of Omar Khayyam

{ewl msdn cd.dll, ewcright, /c"Microsoft"}

Documentation Comments

The view that documentation is something that is added to a program after it has been commissioned seems to be wrong in principle, and counterproductive in practice. Instead, documentation must be regarded as an integral part of the process of design and coding.

--C. A. R. Hoare, Hints on Programming Language Design (1973)

Java programs can include documentation in their source code, in special documentation comments ([§3.7](#)). Such comments can appear before each class or interface declaration and before each method, constructor, or field declaration. Hypertext web pages can then be produced automatically from the source code of the program and these documentation comments.

This chapter gives an informal description of documentation comments. A complete formal specification would require a detailed description of those parts of the Hypertext Markup Language (HTML) that can be used within the documentation comments, which is beyond the scope of this specification.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.1 The Text of a Documentation Comment

The text of a documentation comment consists of the characters between the `/**` that begins the comment and the `*/` that ends it. The text is divided into one or more lines. On each of these lines, leading `*` characters are ignored; for lines other than the first, blanks and tabs preceding the initial `*` characters are also discarded.

So, for example, in the comment:

```
/**XYZ  
  ** Initialize to pre-trial defaults.  
  123*/
```

the text of the comment has three lines. The first line consists of the text "XYZ"; the second line consists of the text " Initialize to pre-trial defaults." and the third line consists of the text "123"

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.2 HTML in a Documentation Comment

Text in a documentation comment may use HTML markers for formatting, with the exception that the specific markers `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>`, `<H6>`, and `<HR>` are reserved for use by the documentation generator and should not be used in the text. A complete description of HTML is available from the web site <http://www.w3.org> and also through the Internet documentation database at <http://www.internic.net>, where the document "Hypertext Markup Language -Version 2.0" by T. Berners-Lee and D. Connolly may be found as RFC1866.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.3 Summary Sentence and General Description

The first sentence of each documentation comment should be a summary sentence, containing a concise but complete description of the declared entity. This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tagline (as defined below). This simple rule means that a first sentence such as:

```
This is a simulation of Prof. Knuth's MIX computer.
```

will not work properly, because the period after the abbreviation "Prof" ends the first sentence, as far as the Java documentation comment processor is concerned. Take care to avoid such difficulties.

Sentences following the summary sentence but preceding the first tagged paragraph (if any) form the general description part of the documentation comment.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4 Tagged Paragraphs

A line of a documentation comment that begins with the character @ followed by one of a few special keywords starts a *tagged paragraph*. The tagged paragraph also includes any following lines up to, but not including, either the first line of the next tagged paragraph or the end of the documentation comment.

Tagged paragraphs identify certain information that has a routine structure, such as the intended purpose of each parameter of a method, in a form that the documentation comment processor can easily marshal into standard typographical formats for purposes of presentation and cross-reference.

Different kinds of tagged paragraphs are available for class and interface declarations and for method, field, and constructor declarations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.1 The @see Tag

The following are examples of @see paragraphs, which may be used in any documentation comment to indicate a cross-reference to a class, interface, method, constructor, field, or URL:

```
@see java.lang.String
@see String
@see java.io.InputStream;
@see String#equals
@see java.lang.Object#wait(int)
@see java.io.RandomAccessFile#RandomAccessFile(File, String)
@see Character#MAX_RADIX
@see <a href="spec.html">Java Spec</a>
```

The character # separates the name of a class from the name of one of its fields, methods, or constructors. One of several overloaded methods or constructors may be selected by including a parenthesized list of argument types after the method or constructor name.

A documentation comment may contain more than one @see tag.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.2 The @author Tag

The following are examples of @author taglines, which may be used in documentation comments for class and interface declarations:

```
@author Mary Wollstonecraft
@author Hildegard von Bingen
@author Dorothy Parker
```

The information in an @author paragraph has no special internal structure.

A documentation comment may contain more than one @author tag. Alternatively, a single @author paragraph may mention several authors:

```
@author Jack Kent, Peggy Parish, Crockett Johnson,
_James Marshall, Marjorie Weinman Sharmat,
_Robert McCloskey, and Madeleine L'Engle
```

However, we recommend specifying one author per @author paragraph, which allows the documentation processing tool to provide the correct punctuation in all circumstances.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.3 The @version Tag

The following is an example of a `@version` paragraph, which may be used in documentation comments for class and interface declarations:

```
@version 493.0.1beta
```

The information in a `@version` paragraph has no special internal structure.

A documentation comment may contain at most one `@version` tag.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.4 The @param Tag

The following are examples of @param paragraphs, which may be used in documentation comments for method and constructor declarations:

```
@param file the file to be searched
@param pattern
    _the pattern to be matched during the search
@param count      the number of lines to print for each match
```

The information in a @param paragraph should consist of the name of the parameter followed by a short description.

A documentation comment may contain more than one @param tag. The usual convention is that if any @param paragraphs are present in a documentation comment, then there should be one @param paragraph for each parameter of the method or constructor, and the @param paragraphs should appear in the order in which the parameters are declared.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.5 The @return Tag

The following is an example of a `@return` paragraph, which may be used in documentation comments for declarations of methods whose result type is not `void`:

```
@return the number of widgets that pass the quality test
```

The information in a `@return` paragraph has no special internal structure. The usual convention is that it consists of a short description of the returned value.

A documentation comment may contain at most one `@return` tag.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.4.6 The @exception Tag

The following is an example of an `@exception` paragraph, which may be used in documentation comments for method and constructor declarations:

```
@exception IndexOutOfBoundsException  
____ the matrix is too large  
@exception UnflangedWidgetException the widget does not  
____ have a flange, or its flange has size zero  
@exception java.io.FileNotFoundException the file  
____ does not exist
```

The information in an `@exception` paragraph should consist of the name of an exception class (which may be a simple name or a qualified name) followed by a short description of the circumstances that cause the exception to be thrown.

A documentation comment may contain more than one `@exception` tag.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


18.5 Example

Here, as an example, is a version of the source code for the class `Object` of the package `java.lang`, including its documentation comments.

```
/*
 * @(#)Object.java 1.37 96/06/26
 *
 * Copyright (c) 1994, 1995, 1996 Sun Microsystems, Inc.
 * All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for NON-COMMERCIAL purposes
 * and without fee is hereby granted provided that this
 * copyright notice appears in all copies. Please refer to
 * the file "copyright.html" for further important copyright
 * and licensing information.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE
 * SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR
 * NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
 * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
 */

package java.lang;

/**
 * The root of the Class hierarchy. Every Class in the
 * system has Object as its ultimate parent. Every variable
 * and method defined here is available in every Object.
 * @see ____Class
 * @version____1.37, 26 Jun 1996
 */

public class Object {
    /**
     * Returns the Class of this Object. Java has a runtime
     * representation for classes--a descriptor of type Class
     * --which the method getClass() returns for any Object.
     */
    public final native Class getClass();

    /**
     * Returns a hashCode for this Object.
     * Each Object in the Java system has a hashCode.
     * The hashCode is a number that is usually different

```



```

_  * for different Objects. It is used when storing Objects
_  * in hashtables.
_  * Note: hashcodes can be negative as well as positive.
_  * @see ____java.util.Hashtable
_  */
_ public native int hashCode();

_
_ /**
_  * Compares two Objects for equality.
_  * Returns a boolean that indicates whether this Object
_  * is equivalent to the specified Object. This method is
_  * used when an Object is stored in a hashtable.
_  * @param ____obj____ the Object to compare with
_  * @return ____true if these Objects are equal;
_  * ____false otherwise.
_  * @see ____java.util.Hashtable
_  */
_ public boolean equals(Object obj) {
_     return (this == obj);
_ }

_
_ /**
_  * Creates a clone of the object. A new instance is
_  * allocated and a bitwise clone of the current object
_  * is placed in the new object.
_  * @return ____a clone of this Object.
_  * @exception ____OutOfMemoryError If there is not enough
_  * ____memory.
_  * @exception ____CloneNotSupportedException Object
_  * ____explicitly does not want to be
_  * ____cloned, or it does not support
_  * ____the Cloneable interface.
_  */
_ protected native Object clone()
_     throws CloneNotSupportedException;

_
_
_
_ /**
_  * Returns a String that represents this Object.
_  * It is recommended that all subclasses override
_  * this method.
_  */
_ public String toString() {
_     return getClass().getName() + "@" +
_     Integer.toHexString(hashCode());
_ }

```



```

_ /**
_  * Notifies a single waiting thread on a change in
_  * condition of another thread. The thread effecting
_  * the change notifies the waiting thread using notify().
_  * Threads that want to wait for a condition to change
_  * before proceeding can call wait(). <p>
_  * <em>The method notify() can be called only by the
_  * thread that is the owner of the current object's
_  * monitor lock.</em>
_  *
_  * @exception _____ IllegalMonitorStateException If the
_  * _____ current thread is not the owner
_  * _____ of the Object's monitor lock.
_  * @see _____ Object#wait
_  * @see _____ Object#notifyAll
_  */
_ public final native void notify();

_ /**
_  * Notifies all the threads waiting for a condition to
_  * change. Threads that are waiting are generally waiting
_  * for another thread to change some condition. Thus, the
_  * thread effecting a change that more than one thread is
_  * waiting for notifies all the waiting threads using
_  * the method notifyAll(). Threads that want to wait for
_  * a condition to change before proceeding can call
_  * wait(). <p>
_  * <em>The method notifyAll() can be called only by the
_  * thread that is the owner of the current object's
_  * monitor lock.</em>
_  *
_  * @exception _____ IllegalMonitorStateException If the
_  * _____ current thread is not the owner
_  * _____ of the Object's monitor lock.
_  * @see _____ Object#wait
_  * @see _____ Object#notify
_  */
_ public final native void notifyAll();

_ /**
_  * Causes a thread to wait until it is notified or the
_  * specified timeout expires. <p>
_  * <em>The method wait(millis) can be called only by
_  * the thread that is the owner of the current object's
_  * monitor lock.</em>
_  *
_  * @param _____ millis_____ the maximum time to wait,
_  * _____ in milliseconds
_  * @exception _____ IllegalMonitorStateException If the
_  * _____ current thread is not the owner
_  * _____ of the Object's monitor lock.
_  * @exception _____ InterruptedException Another thread has

```



```

- * _____interrupted this thread.
- */
- public final native void wait(long millis)
- throws InterruptedException;

- /**
- * More accurate wait.
- * <em>The method wait(millis, nanos) can be called only
- * by the thread that is the owner of the current
- * object's monitor lock.</em>
- *
- * @param millis_____the maximum time to wait,
- * _____in milliseconds
- * @param nano_____additional time to wait,
- * _____in nanoseconds
- * _____(range 0-999999)
- * @exception_____IllegalMonitorStateException If the
- * _____current thread is not the owner
- * _____of the Object's monitor lock.
- * @exception _____InterruptedException Another thread has
- * _____interrupted this thread.
- */
- public final void wait(long millis, int nanos)
- throws InterruptedException
- {
- if (nanos >= 500000 || (nanos != 0 && millis==0))
- timeout++;
- wait(timeout);
- }

- /**
- * Causes a thread to wait forever until it is notified.
- * <p>
- * <em>The method wait() can be called only by the
- * thread that is the owner of the current object's
- * monitor lock.</em>
- *
- *
- * @exception_____IllegalMonitorStateException If the
- * _____current thread is not the owner
- * _____of the Object's monitor lock.
- * @exception _____InterruptedException Another thread has
- * _____interrupted this thread.
- */
- public final void wait() throws InterruptedException {
- wait(0);
- }

```



```

_ /**
_  * Code to perform when this object is garbage collected.
_  * The default is that nothing needs to be performed.
_  *
_  * Any exception thrown by a finalize method causes the
_  * finalization to halt. But otherwise, it is ignored.
_  */
_ protected void finalize() throws Throwable { }
_
_ }

```

From this source code, the javadoc tool produced the following HTML file, which is available for browsing at <http://java.sun.com/Series>, our Java Series web site:

```
<!--NewPage-->
```

```
<html>
```

```
<head>
```

```
<!-- Generated by javadoc on Wed Jun 26 11:40:38 EDT 1996 -->
```

```
<a name="_top_"></a>
```

```
<title>
```

```
Class java.lang.Object
```

```
</title>
```

```
</head>
```

```
<body>
```

```
<pre>
```

```

<a href="packages.html">All Packages</a> <a href="tree.html">Class Hierarchy<
</a> <a href="Package-java.lang.html">This Package</a> <a href="java.lang.Nu
umber.html#_top_">Previous</a> <a href="java.lang.OutOfMemoryError.html#_top_
">Next</a> <a href="AllNames.html">Index</a></pre>

```


<hr>

<h1>

Class java.lang.Object

</h1>

<pre>

java.lang.Object

</pre>

<hr>

<dl>

<dt> public class Object

</dl>

The root of the Class hierarchy. Every Class in the

system has Object as its ultimate parent. Every variable

and method defined here is available in every Object.

<dl>

<dt> Version:

<dd> 1.37, 26 Jun 1996

<dt> See Also:

<dd> Class

</dl>

<hr>

<h2>

</h2>

<dl>

<dt>

Object()

<dd>

</dl>

<h2>

</h2>

<dl>

<dt>

clone()

<dd> Creates a clone of the object.

<dt>

equals(Object)

<dd> Compares two Objects for equality.

<dt>

finalize()

<dd> Code to perform when this object is garbage collected.

<dt>

getClass()

<dd> Returns the Class of this Object.

<dt>

hashCode()

<dd> Returns a hashcode for this Object.

<dt>

notify()

<dd> Notifies a single waiting thread on a change in

condition of another thread.

<dt>


[notifyAll\(\)](#notifyAll())

<dd> Notifies all the threads waiting for a condition to change.

<dt>  >

[toString\(\)](#toString())

<dd> Returns a String that represents this Object.

<dt>  >

[wait\(\)](#wait())

<dd> Causes a thread to wait forever until it is notified.

<dt>  >

[wait\(long\)](#wait(long))

<dd> Causes a thread to wait until it is notified or the specified timeout expires.

<dt>  >

[wait\(long, int\)](#wait(long, int))

<dd> More accurate wait.

</dl>

[constructors](#)

<h2>

</h2>

Object

<pre>

public Object()

</pre>

<h2>

</h2>

getClass

<pre>

public final Class getClass()

</pre>

<dl>

<dd> Returns the Class of this Object. Java has a runtime

representation for classes--a descriptor of type Class

--which the method getClass() returns for any Object.

</dl>

hashCode

<pre>

public int hashCode()

</pre>

<dl>

<dd> Returns a hashcode for this Object.

Each Object in the Java system has a hashcode.

The hashcode is a number that is usually different

for different Objects. It is used when storing Objects

in hashtables.

Note: hashcodes can be negative as well as positive.

<dl>

<dt> See Also:

<dd> Hashtable

</dl>

</dl>

equals

<pre>

public boolean equals(Object obj)

</pre>

<dl>

<dd> Compares two Objects for equality.

Returns a boolean that indicates whether this Object

is equivalent to the specified Object. This method is

used when an Object is stored in a hashtable.

<dl>

<dt> Parameters:

<dd> obj - the Object to compare with

<dt> Returns:

<dd> true if these Objects are equal;

false otherwise.

<dt> See Also:

<dd> Hashtable

</dl>

</dl>

clone

<pre>

protected Object clone() throws CloneNotSupportedException

</pre>

<dl>

<dd> Creates a clone of the object. A new instance is

allocated and a bitwise clone of the current object

is placed in the new object.

<dl>

<dt> Returns:

<dd> a clone of this Object.

<dt> Throws: OutOfMemoryError

<dd> If there is not enough

memory.

<dt> Throws: CloneNotSupportedException

<dd> Object

explicitly does not want to be

cloned, or it does not support

the Cloneable interface.

</dl>

</dl>

toString

<pre>

public String toString()

</pre>

<dl>

<dd> Returns a String that represents this Object.

It is recommended that all subclasses override

this method.

</dl>

notify

<pre>

public final void notify()

</pre>

<dl>

<dd> Notifies a single waiting thread on a change in

condition of another thread. The thread effecting

the change notifies the waiting thread using notify().

Threads that want to wait for a condition to change

before proceeding can call wait(). <p>

The method notify() can be called only by the

thread that is the owner of the current object's

monitor lock.

<dl>

<dt> Throws: IllegalMonitorStateException

<dd> If the

current thread is not the owner

of the Object's monitor lock.

<dt> See Also:

<dd> wait, notifyAll

</dl>

</dl>

notifyAll

<pre>

public final void notifyAll()

</pre>

<dl>

<dd> Notifies all the threads waiting for a condition to

change. Threads that are waiting are generally waiting

for another thread to change some condition. Thus, the

thread effecting a change that more than one thread is

waiting for notifies all the waiting threads using

the method notifyAll(). Threads that want to wait for

a condition to change before proceeding can call

wait(). <p>

The method notifyAll() can be called only by the

thread that is the owner of the current object's

monitor lock.

<dl>

<dt> Throws: IllegalMonitorStateException

<dd> If the

current thread is not the owner

of the Object's monitor lock.

<dt> See Also:

<dd> wait, notify

</dl>

</dl>

 o ">

wait

<pre>

```
public final void wait(long millis) throws <a href="java.lang.InterruptedException.html#_top_">InterruptedException</a>
```

</pre>

<dl>

<dd> Causes a thread to wait until it is notified or the

specified timeout expires. <p>

The method wait(millis) can be called only by

the thread that is the owner of the current object's

monitor lock.

<dl>

<dt> Parameters:

<dd> millis - the maximum time to wait,

in milliseconds

<dt> Throws: IllegalMonitorStateException

<dd> If the

current thread is not the owner

of the Object's monitor lock.

<dt> Throws: InterruptedException

<dd> Another thread has

interrupted this thread.

</dl>

</dl>

wait

<pre>

public final void wait(long millis,

int nanos) throws InterruptedException

</pre>

<dl>

<dd> More accurate wait.

The method wait(millis, nanos) can be called only

by the thread that is the owner of the current

object's monitor lock.

<dl>

<dt> Parameters:

<dd> millis - the maximum time to wait,

in milliseconds

<dd> nano - additional time to wait,

in nanoseconds

(range 0-999999)

<dt> Throws: IllegalMonitorStateException

<dd> If the

current thread is not the owner

of the Object's monitor lock.

<dt> Throws: InterruptedException

<dd> Another thread has

interrupted this thread.

</dl>

</dl>

wait

<pre>

```
public final void wait() throws <a href="java.lang.InterruptedException.html#_top_">InterruptedException</a>
```

</pre>

<dl>

<dd> Causes a thread to wait forever until it is notified.

<p>

The method wait() can be called only by the

thread that is the owner of the current object's

monitor lock.

<dl>

<dt> Throws: IllegalMonitorStateException

<dd> If the

current thread is not the owner

of the Object's monitor lock.

<dt> Throws: InterruptedException

<dd> Another thread has

interrupted this thread.

</dl>

</dl>

finalize

<pre>

protected void finalize() throws Throwable

</pre>

<dl>

<dd> Code to perform when this object is garbage collected.

The default is that nothing needs to be performed.

Any exception thrown by a finalize method causes the

finalization to halt. But otherwise, it is ignored.

```
</dl>
```

```
<hr>
```

```
<pre>
```

```
<a href="packages.html">All Packages</a> <a href="tree.html">Class Hierarchy↵
</a> <a href="Package-java.lang.html">This Package</a> <a href="java.lang.N↵
umber.html#_top_">Previous</a> <a href="java.lang.OutOfMemoryError.html#_top↵
">Next</a> <a href="AllNames.html">Index</a></pre>
```

```
</body>
```

```
</html>
```

Many of the lines in this HTML file are far too long to fit onto these pages. We have used the character "↵" at the end of a line to indicate that the following line of text on the page is part of the same line in the generated file.

This generated HTML file is meant only as an example, not as a specification of the behavior of the `javadoc` tool, which may be changed over time to improve the HTML presentation of the documentation information.

*Very few facts are able to tell their own story,
without comments to bring out their meaning.*
--John Stuart Mill, *On Liberty* (1869)

LALR(1) Grammar

This chapter presents a grammar for Java. The grammar has been mechanically checked to insure that it is LALR(1).

The grammar for Java presented piecemeal in the preceding chapters is much better for exposition, but it cannot be parsed left-to-right with one token of lookahead because of certain syntactic peculiarities, some of them inherited from C and C++. These problems and the solutions adopted for the LALR(1) grammar are presented below, followed by the grammar itself.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1 Grammatical Difficulties

There are five problems with the grammar presented in preceding chapters.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1.1 Problem #1: Names Too Specific

Consider the two groups of productions:

PackageName:

Identifier

PackageName . Identifier

TypeName:

Identifier

PackageName . Identifier

and:

MethodName:

Identifier

AmbiguousName . Identifier

AmbiguousName:

Identifier

AmbiguousName . Identifier

Now consider the partial input:

```
class Problem1 { int m() { hayden.
```

When the parser is considering the token `hayden`, with one-token lookahead to symbol `"."`, it cannot yet tell whether `hayden` should be a *PackageName* that qualifies a type name, as in:

```
hayden.Dinosaur rex = new Hayden.Dinosaur(2);
```

or an *AmbiguousName* that qualifies a method name, as in:

```
hayden.print("Dinosaur Rex!");
```

Therefore, the productions shown above result in a grammar that is not LALR(1). There are also other problems with drawing distinctions among different kinds of names in the grammar.

The solution is to eliminate the nonterminals *PackageName*, *TypeName*, *ExpressionName*, *MethodName*, and *AmbiguousName*, replacing them all with a single nonterminal *Name*:

Name:

SimpleName

QualifiedName

SimpleName:

Identifier

QualifiedName:

Name . *Identifier*

A later stage of compiler analysis then sorts out the precise role of each name or name qualifier.

For related reasons, these productions in §4.3:

ClassOrInterfaceType:

ClassType

InterfaceType

ClassType:

TypeName

InterfaceType:

TypeName

were changed to:

ClassOrInterfaceType:

Name

ClassType:

ClassOrInterfaceType

InterfaceType:

ClassOrInterfaceType


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1.2 Problem #2: Modifiers Too Specific

Consider the two groups of productions:

FieldDeclaration:

FieldModifiersopt Type VariableDeclarators ;

FieldModifiers:

FieldModifier

FieldModifiers FieldModifier

FieldModifier: one of

public protected private

final static transient volatile

and:

MethodHeader:

MethodModifiersopt ResultType MethodDeclarator Throwsopt

MethodModifiers:

MethodModifier

MethodModifiers MethodModifier

MethodModifier: one of

public protected private

static

abstract final native synchronized

Now consider the partial input:

```
class Problem2 { public static int
```

When the parser is considering the token `static`, with one-token lookahead to symbol `int`-or, worse yet, considering the token `public` with lookahead to `static`-it cannot yet tell whether this will be a field declaration such as:


```
public static int maddie = 0;
```

or a method declaration such as:

```
public static int maddie(String art) { return art.length(); }
```

Therefore, the parser cannot tell with only one-token lookahead whether `static` (or, similarly, `public`) should be reduced to *FieldModifier* or *MethodModifier*. Therefore, the productions shown above result in a grammar that is not LALR(1). There are also other problems with drawing distinctions among different kinds of modifiers in the grammar.

While not all contexts provoke the problem, the simplest solution is to combine all contexts in which such modifiers are used, eliminating all six of the nonterminals *ClassModifiers* (§8.1.2), *FieldModifiers* (§8.3.1), *MethodModifiers* (§8.4.3), *ConstructorModifiers* (§8.6.3), *InterfaceModifiers* (§9.1.2), and *ConstantModifiers* (§9.3) from the grammar, replacing them all with a single nonterminal *Modifiers*:

Modifiers:

Modifier

Modifiers Modifier

Modifier: one of

public protected private

static

abstract final native synchronized transient volatile

A later stage of compiler analysis then sorts out the precise role of each modifier and whether it is permitted in a given context.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1.3 Problem #3: Field Declaration versus Method Declaration

Consider the two productions (shown after problem #2 has been corrected):

FieldDeclaration:

Modifiersopt Type VariableDeclarators ;

and:

MethodHeader:

Modifiersopt ResultType MethodDeclarator Throwsopt

where *ResultType* is defined as:

ResultType:

Type

void

Now consider the partial input:

```
class Problem3 { int julie
```

Note that, in this simple example, no *Modifiers* are present. When the parser is considering the token *int*, with one-token lookahead to symbol *julie*, it cannot yet tell whether this will be a field declaration such as:

```
int julie = 14;
```

or a method declaration such as:

```
int julie(String art) { return art.length(); }
```

Therefore, after the parser reduces *int* to the nonterminal *Type*, it cannot tell with only one-token lookahead whether *Type* should be further reduced to *ResultType* (for a method declaration) or left alone (for a field declaration). Therefore, the productions shown above result in a grammar that is not LALR(1).

The solution is to eliminate the *ResultType* production and to have separate alternatives for *MethodHeader*:

MethodHeader:

Modifiersopt Type MethodDeclarator Throwsopt

Modifiersopt void MethodDeclarator Throwsopt

This allows the parser to reduce `int` to *Type* and then leave it as is, delaying the decision as to whether a field declaration or method declaration is in progress.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1.4 Problem #4: Array Type versus Array Access

Consider the productions (shown after problem #1 has been corrected):

ArrayType:

Type []

and:

ArrayAccess:

Name [*Expression*]

PrimaryNoNewArray [*Expression*]

Now consider the partial input:

```
class Problem4 { Problem4() { peter[
```

When the parser is considering the token `peter`, with one-token lookahead to symbol `[`, it cannot yet tell whether `peter` will be part of a type name, as in:

```
peter[] team;
```

or part of an array access, as in:

```
peter[3] = 12;
```

Therefore, after the parser reduces `peter` to the nonterminal *Name*, it cannot tell with only one-token lookahead whether *Name* should be reduced ultimately to *Type* (for an array type) or left alone (for an array access). Therefore, the productions shown above result in a grammar that is not LALR(1).

The solution is to have separate alternatives for *ArrayType*:

ArrayType:

PrimitiveType []

Name []

ArrayType []

This allows the parser to reduce `peter` to *Name* and then leave it as is, delaying the decision as to whether an array type or array access is in progress.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.1.5 Problem #5: Cast versus Parenthesized Expression

Consider the production:

CastExpression:

```
( PrimitiveType ) UnaryExpression
( ReferenceType ) UnaryExpressionNotPlusMinus
```

Now consider the partial input:

```
class Problem5 { Problem5() { super((matthew)
```

When the parser is considering the token `matthew`, with one-token lookahead to symbol `)`, it cannot yet tell whether `(matthew)` will be a parenthesized expression, as in:

```
super((matthew), 9);
```

or a cast, as in:

```
super((matthew)baz, 9);
```

Therefore, after the parser reduces `matthew` to the nonterminal *Name*, it cannot tell with only one-token lookahead whether *Name* should be further reduced to *PostfixExpression* and ultimately to *Expression* (for a parenthesized expression) or to *ClassOrInterfaceType* and then to *ReferenceType* (for a cast). Therefore, the productions shown above result in a grammar that is not LALR(1).

The solution is to eliminate the use of the nonterminal *ReferenceType* in the definition of *CastExpression*, which requires some reworking of both alternatives to avoid other ambiguities:

CastExpression:

```
( PrimitiveType Dimsopt ) UnaryExpression
( Expression ) UnaryExpressionNotPlusMinus
( Name Dims ) UnaryExpressionNotPlusMinus
```

This allows the parser to reduce `matthew` to *Expression* and then leave it there, delaying the decision as to whether a parenthesized expression or a cast is in progress. Inappropriate variants such as:

```
(int[])+3
```


and:

```
(matthew+1) baz
```

must then be weeded out and rejected by a later stage of compiler analysis.

The remaining sections of this chapter constitute a LALR(1) grammar for Java syntax, in which the five problems described above have been solved.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.2 Grammar from §2.3: The Syntactic Grammar §2.3

Goal:

CompilationUnit

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.3 Grammar from §3: Lexical Structure §3

Literal:

IntegerLiteral

FloatingPointLiteral

BooleanLiteral

CharacterLiteral

StringLiteral

NullLiteral

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.4 Grammar from §4: Types, Values, and Variables §4

Type:

PrimitiveType

ReferenceType

PrimitiveType:

NumericType

boolean

NumericType:

IntegralType

FloatingPointType

IntegralType: one of

byte short int long char

FloatingPointType: one of

float double

ReferenceType:

ClassOrInterfaceType

ArrayType

ClassOrInterfaceType:

Name

ClassType:

ClassOrInterfaceType

InterfaceType:

ClassOrInterfaceType

ArrayType:

PrimitiveType []

Name []

ArrayType []


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.5 Grammar from §6: Names §6

Name:

SimpleName

QualifiedName

SimpleName:

Identifier

QualifiedName:

Name . Identifier

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.6 Grammar from §7: Packages §7

CompilationUnit:

PackageDeclarationopt ImportDeclarationsopt TypeDeclarationsopt

ImportDeclarations:

ImportDeclaration

ImportDeclarations ImportDeclaration

TypeDeclarations:

TypeDeclaration

TypeDeclarations TypeDeclaration

PackageDeclaration:

package Name ;

ImportDeclaration:

SingleTypeImportDeclaration

TypeImportOnDemandDeclaration

SingleTypeImportDeclaration:

import Name ;

TypeImportOnDemandDeclaration:

*import Name . * ;*

TypeDeclaration:

ClassDeclaration

InterfaceDeclaration

;

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.7 Productions Used Only in the LALR(1) Grammar

Modifiers:

Modifier

Modifiers Modifier

Modifier: one of

public protected private

static

abstract final native synchronized transient volatile

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.8 Grammar from §8: Classes §8

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.8.1 Grammar from §8.1: Class Declaration §8.1

ClassDeclaration:

Modifiersopt class Identifier Superopt Interfacesopt ClassBody

Super:

extends ClassType

Interfaces:

implements InterfaceTypeList

InterfaceTypeList:

InterfaceType

InterfaceTypeList , InterfaceType

ClassBody:

{ ClassBodyDeclarationsopt }

ClassBodyDeclarations:

ClassBodyDeclaration

ClassBodyDeclarations ClassBodyDeclaration

ClassBodyDeclaration:

ClassMemberDeclaration

StaticInitializer

ConstructorDeclaration

ClassMemberDeclaration:

FieldDeclaration

MethodDeclaration

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.8.2 Grammar from §8.3: Field Declarations §8.3

FieldDeclaration:

Modifiersopt Type VariableDeclarators ;

VariableDeclarators:

VariableDeclarator

VariableDeclarators , VariableDeclarator

VariableDeclarator:

VariableDeclaratorId

VariableDeclaratorId = VariableInitializer

VariableDeclaratorId:

Identifier

VariableDeclaratorId []

VariableInitializer:

Expression

ArrayInitializer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.8.3 Grammar from §8.4: Method Declarations §8.4

MethodDeclaration:

MethodHeader MethodBody

MethodHeader:

Modifiersopt Type MethodDeclarator Throwsopt

Modifiersopt void MethodDeclarator Throwsopt

MethodDeclarator:

Identifier (FormalParameterListopt)

MethodDeclarator []

FormalParameterList:

FormalParameter

FormalParameterList , FormalParameter

FormalParameter:

Type VariableDeclaratorId

Throws:

throws ClassTypeList

ClassTypeList:

ClassType

ClassTypeList , ClassType

MethodBody:

Block

;

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.8.4 Grammar from §8.5: Static Initializers §8.5

StaticInitializer:

static Block

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.8.5 Grammar from §8.6: Constructor Declarations §8.6

ConstructorDeclaration:

Modifiersopt ConstructorDeclarator Throwsopt ConstructorBody

ConstructorDeclarator:

SimpleName (FormalParameterListopt)

ConstructorBody:

{ ExplicitConstructorInvocationopt BlockStatementsopt }

ExplicitConstructorInvocation:

this (ArgumentListopt) ;

super (ArgumentListopt) ;

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.9 Grammar from §9: Interfaces §9

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


19.9.1 Grammar from §9.1: Interface Declarations §9.1

InterfaceDeclaration:

Modifiersopt interface Identifier ExtendsInterfacesopt InterfaceBody

ExtendsInterfaces:

extends InterfaceType

ExtendsInterfaces , InterfaceType

InterfaceBody:

{ InterfaceMemberDeclarationsopt }

InterfaceMemberDeclarations:

InterfaceMemberDeclaration

InterfaceMemberDeclarations InterfaceMemberDeclaration

InterfaceMemberDeclaration:

ConstantDeclaration

AbstractMethodDeclaration

ConstantDeclaration:

FieldDeclaration

AbstractMethodDeclaration:

MethodHeader ;

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.10 Grammar from §10: Arrays §10

ArrayInitializer:

{ VariableInitializersopt ,opt }

VariableInitializers:

VariableInitializer

VariableInitializers , VariableInitializer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.11 Grammar from §14: Blocks and Statements §14

Block:

{ BlockStatementsopt }

BlockStatements:

BlockStatement

BlockStatements BlockStatement

BlockStatement:

LocalVariableDeclarationStatement

Statement

LocalVariableDeclarationStatement:

LocalVariableDeclaration ;

LocalVariableDeclaration:

Type VariableDeclarators

Statement:

StatementWithoutTrailingSubstatement

LabeledStatement

IfThenStatement

IfThenElseStatement

WhileStatement

ForStatement

StatementNoShortIf:

StatementWithoutTrailingSubstatement

LabeledStatementNoShortIf

IfThenElseStatementNoShortIf

WhileStatementNoShortIf

ForStatementNoShortIf

StatementWithoutTrailingSubstatement:

Block

EmptyStatement

ExpressionStatement

SwitchStatement

DoStatement

BreakStatement

ContinueStatement

ReturnStatement

SynchronizedStatement

ThrowStatement

TryStatement

EmptyStatement:

;

LabeledStatement:

Identifier : Statement

LabeledStatementNoShortIf:

Identifier : StatementNoShortIf

ExpressionStatement:

StatementExpression ;

StatementExpression:

Assignment

PreIncrementExpression

PreDecrementExpression

PostIncrementExpression

PostDecrementExpression

MethodInvocation

ClassInstanceCreationExpression

IfThenStatement:

if (Expression) Statement

IfThenElseStatement:

if (Expression) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:

if (Expression) StatementNoShortIf else StatementNoShortIf

SwitchStatement:

switch (Expression) SwitchBlock

SwitchBlock:

{ SwitchBlockStatementGroupsopt SwitchLabelsopt }

SwitchBlockStatementGroups:

SwitchBlockStatementGroup

SwitchBlockStatementGroups SwitchBlockStatementGroup

SwitchBlockStatementGroup:

SwitchLabels BlockStatements

SwitchLabels:

SwitchLabel

SwitchLabels SwitchLabel

SwitchLabel:

case ConstantExpression:

default :

WhileStatement:

while (Expression) Statement

WhileStatementNoShortIf:

while (Expression) StatementNoShortIf

DoStatement:

do Statement while (Expression) ;

ForStatement:

*for (ForInitopt ; Expressionopt ; ForUpdateopt)
Statement*

ForStatementNoShortIf:

*for (ForInitopt ; Expressionopt ; ForUpdateopt)
StatementNoShortIf*

ForInit:

*StatementExpressionList
LocalVariableDeclaration*

ForUpdate:

StatementExpressionList

StatementExpressionList:

*StatementExpression
StatementExpressionList , StatementExpression*

BreakStatement:

break Identifieropt ;

ContinueStatement:

continue Identifieropt ;

ReturnStatement:

return Expressionopt ;

ThrowStatement:

throw Expression ;

SynchronizedStatement:

synchronized (Expression) Block

TryStatement:

*try Block Catches
try Block Catchesopt Finally*

Catches:

CatchClause

Catches CatchClause

CatchClause:

catch (FormalParameter) Block

Finally:

finally Block

{ewl msdncd.dll, ewcright, /c"Microsoft"}

19.12 Grammar from §15: Expressions §15

Primary:

PrimaryNoNewArray

ArrayCreationExpression

PrimaryNoNewArray:

Literal

this

(Expression)

ClassInstanceCreationExpression

FieldAccess

MethodInvocation

ArrayAccess

ClassInstanceCreationExpression:

new ClassType (ArgumentListopt)

ArgumentList:

Expression

ArgumentList , Expression

ArrayCreationExpression:

new PrimitiveType DimExprs Dimsopt

new ClassOrInterfaceType DimExprs Dimsopt

DimExprs:

DimExpr

DimExprs DimExpr

DimExpr:

[Expression]

Dims:

[]

Dims []

FieldAccess:

Primary . Identifier

super . Identifier

MethodInvocation:

Name (ArgumentListopt)

Primary . Identifier (ArgumentListopt)

super . Identifier (ArgumentListopt)

ArrayAccess:

Name [Expression]

PrimaryNoNewArray [Expression]

PostfixExpression:

Primary

Name

PostIncrementExpression

PostDecrementExpression

PostIncrementExpression:

PostfixExpression ++

PostDecrementExpression:

PostfixExpression --

UnaryExpression:

PreIncrementExpression

PreDecrementExpression

+ UnaryExpression

- UnaryExpression

UnaryExpressionNotPlusMinus

PreIncrementExpression:


```

    ++ UnaryExpression
PreDecrementExpression:
    -- UnaryExpression
UnaryExpressionNotPlusMinus:
    PostfixExpression
    ~ UnaryExpression
    ! UnaryExpression
    CastExpression
CastExpression:
    ( PrimitiveType Dimsopt ) UnaryExpression
    ( Expression ) UnaryExpressionNotPlusMinus
    ( Name Dims ) UnaryExpressionNotPlusMinus
MultiplicativeExpression:
    UnaryExpression
    MultiplicativeExpression * UnaryExpression
    MultiplicativeExpression / UnaryExpression
    MultiplicativeExpression % UnaryExpression
AdditiveExpression:
    MultiplicativeExpression
    AdditiveExpression + MultiplicativeExpression
    AdditiveExpression - MultiplicativeExpression
ShiftExpression:
    AdditiveExpression
    ShiftExpression << AdditiveExpression
    ShiftExpression >> AdditiveExpression
    ShiftExpression >>> AdditiveExpression
RelationalExpression:
    ShiftExpression
    RelationalExpression < ShiftExpression

```


RelationalExpression > ShiftExpression

RelationalExpression <= ShiftExpression

RelationalExpression >= ShiftExpression

RelationalExpression instanceof ReferenceType

EqualityExpression:

RelationalExpression

EqualityExpression == RelationalExpression

EqualityExpression != RelationalExpression

AndExpression:

EqualityExpression

AndExpression & EqualityExpression

ExclusiveOrExpression:

AndExpression

ExclusiveOrExpression ^ AndExpression

InclusiveOrExpression:

ExclusiveOrExpression

InclusiveOrExpression | ExclusiveOrExpression

ConditionalAndExpression:

InclusiveOrExpression

ConditionalAndExpression && InclusiveOrExpression

ConditionalOrExpression:

ConditionalAndExpression

ConditionalOrExpression || ConditionalAndExpression

ConditionalExpression:

ConditionalOrExpression

ConditionalOrExpression ? Expression : ConditionalExpression

AssignmentExpression:

ConditionalExpression

Assignment

Assignment:

LeftHandSide AssignmentOperator AssignmentExpression

LeftHandSide:

Name

FieldAccess

ArrayAccess

AssignmentOperator: one of

*= *= /= %= += -= <<= >>= >>>= &= ^= |=*

Expression:

AssignmentExpression

ConstantExpression:

Expression

{ewl msdncd.dll, ewcright, /c"Microsoft"}

The Package java.lang

The `java.lang` package contains classes that are fundamental to the design of the Java language. The most important classes are `Object`, which is the root of the class hierarchy, and `Class`, instances of which represent classes at run time.

Frequently it is necessary to represent a value of primitive type as if it were an object. The wrapper classes `Boolean`, `Character`, `Integer`, `Long`, `Float`, and `Double` serve this purpose. An object of type `Double`, for example, contains a field whose type is `double`, representing that value in such a way that a reference to it can be stored in a variable of reference type. These classes also provide a number of methods for converting among primitive values, as well as supporting such standard methods as `equals` and `hashCode`.

The class `Math` provides commonly used mathematical functions such as sine, cosine, and square root. The classes `String` and `StringBuffer` similarly provide commonly used operations on character strings.

Classes `ClassLoader`, `Process`, `Runtime`, `SecurityManager`, and `System` provide "system operations" that manage the dynamic loading of classes, creation of external processes, host environment inquiries such as the time of day, and enforcement of security policies.

Class `Throwable` encompasses objects that may be thrown by the `throw` statement ([§14.16](#)). Subclasses of `Throwable` represent errors and exceptions.

The hierarchy of classes defined in package `java.lang` is as follows.

```
Object §20.1
interface Cloneable §20.2
    Class §20.3
    Boolean §20.4
    Character §20.5
    Number §20.6
        Integer §20.7
        Long §20.8
        Float §20.9
        Double §20.10
    Math §20.11
    String §20.12
    StringBuffer §20.13
    ClassLoader §20.14
    Process §20.15
    Runtime §20.16
    SecurityManager §20.17
    System §20.18
    interface Runnable §20.19
    Thread §20.20
    ThreadGroup §20.21
    Throwable §20.22
        Error
            LinkageError
                ClassCircularityError
                ClassFormatError
                ExceptionInInitializerError
                IncompatibleClassChangeError
                AbstractMethodError
                IllegalAccessError
```


- InstantiationError
 - NoSuchFieldError
 - NoSuchMethodError
 - NoClassDefFoundError
 - UnsatisfiedLinkError
 - VerifyError
- VirtualMachineError
 - InternalError
 - OutOfMemoryError
 - StackOverflowError
 - UnknownError
- ThreadDeath
- Exception
 - ClassNotFoundException
 - CloneNotSupportedException
 - IllegalAccessException
 - InstantiationException
 - InterruptedException
 - RuntimeException
 - ArithmeticException
 - ArrayStoreException
 - ClassCastException
 - IllegalArgumentException
 - IllegalThreadStateException
 - NumberFormatException
 - IllegalMonitorStateException
 - IndexOutOfBoundsException
 - NegativeArraySizeException
 - NullPointerException
 - SecurityException

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.1 The Class java.lang.Object

The class `Object` is the single root of the class hierarchy. All objects, including arrays, implement the methods of this class.

```
public class Object {
    public final Class getClass();
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    protected Object clone()
        throws CloneNotSupportedException;
    public final void wait()

        throws IllegalMonitorStateException,

        InterruptedException;
    public final void wait(long millis)

        throws IllegalMonitorStateException,

        InterruptedException;
    public final void wait(long millis, int nanos)

        throws IllegalMonitorStateException,
        InterruptedException;
    public final void notify()
        throws IllegalMonitorStateException;
    public final void notifyAll()
        throws IllegalMonitorStateException;
    protected void finalize()

        throws Throwable;
}
```

20.1.1 `public final Class getClass()`

This method returns a reference to the unique object of type `Class` [\(§20.3\)](#) that represents the class of this object. That `Class` object is the object that is locked by `static synchronized` methods of the represented class.

20.1.2 `public String toString()`

The general contract of `toString` is that it returns a string that "textually represents" this object. The idea is to provide a concise but informative representation that will be useful to a person reading it.

The `toString` method defined by class `Object` returns a string consisting of the name of the class of which the object is an instance, a commercial at character '@', and the unsigned hexadecimal representation of the hashCode of the object. In other words, this method returns a string equal to the value of:


```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Overridden by Class ([§20.3](#)), Boolean ([§20.4](#)), Character ([§20.5](#)), Integer ([§20.7](#)), Long ([§20.8](#)), Float ([§20.9](#)), Double ([§20.10](#)), String ([§20.12](#)), StringBuffer ([§20.13](#)), Thread ([§20.20](#)), ThreadGroup ([§20.21](#)), Throwable ([§20.22.4](#)), and Bitset ([§21.2](#)).

20.1.3 `public boolean equals(Object obj)`

This method indicates whether some other object is "equal to" this one.

The general contract of `equals` is that it implements an equivalence relation:

- It is *reflexive*: for any reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used by `x` and `y` in `equals` comparisons is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method defined by class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, `((Object)x).equals(y)` returns `true` if and only if `x` and `y` refer to the same object.

Overridden by Boolean ([§20.4](#)), Character ([§20.5](#)), Integer ([§20.7](#)), Long ([§20.8](#)), Float ([§20.9](#)), Double ([§20.10](#)), String ([§20.12](#)), and Bitset ([§21.2](#)).

20.1.4 `public int hashCode()`

This method is supported principally for the benefit of hash tables such as those provided by the Java library class `java.util.Hashtable` ([§21.5](#)).

The general contract of `hashCode` is as follows:

- Whenever it is invoked on the same object more than once during an execution of a Java application, `hashCode` must consistently return the same integer. The integer may be positive, negative, or zero. This integer does not, however, have to remain consistent from one Java application to another, or from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals` method ([§20.1.3](#)), then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals` method ([§20.1.3](#)), then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the `hashCode` method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java language.)

Overridden by [Boolean \(§20.4\)](#), [Character \(§20.5\)](#), [Integer \(§20.7\)](#), [Long \(§20.8\)](#), [Float \(§20.9\)](#), [Double \(§20.10\)](#), [String \(§20.12\)](#), and [Bitset \(§21.2\)](#).

20.1.5 `protected Object clone()`
throws `CloneNotSupportedException`

The general contract of `clone` is that it creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object `x`, the expression:

```
x.clone() != x
```

will be `true`, and that the expression:

```
x.clone.getClass() == x.getClass()
```

will be `true`, but these are not absolute requirements. While it is typically the case that:

```
x.clone.equals(x)
```

will be `true`, this is not an absolute requirement. Copying an object will typically entail creating a new instance of its class, but it also may require copying of internal data structures as well.

The method `clone` for class `Object` performs a specific cloning operation. First, if the class of this object does not implement the interface `Cloneable`, then a `CloneNotSupportedException` is thrown. Note that all arrays are considered to implement the interface `Cloneable`. Otherwise, this method creates a new instance of the class of this object and initializes all its fields with exactly the contents of the corresponding fields of this object, as if by assignment; the contents of the fields are not themselves cloned. Thus, this method performs a "shallow copy" of this object, not a "deep copy" operation.

The class `Object` does *not* itself implement the interface `Cloneable`, so calling the `clone` method on an object whose class is `Object` will result in throwing an exception at run time. The `clone` method is implemented by the class `Object` as a convenient, general utility for subclasses that implement the interface `Cloneable`, possibly also overriding the `clone` method, in which case the overriding definition can refer to this utility definition by the call:

```
super.clone()
```

20.1.6 `public final void wait()`
throws `IllegalMonitorStateException`, `InterruptedException`

This method causes the current thread to wait until some other thread invokes the `notify` method [\(§20.1.9\)](#) or the `notifyAll` method [\(§20.1.10\)](#) for this object.

In other words, this method behaves exactly as if it simply performs the call `wait(0)` [\(§20.1.7\)](#).

20.1.7 `public final void wait(long millis)`
 throws `IllegalMonitorStateException`, `InterruptedException`

This method causes the current thread to wait until either some other thread invokes the `notify` method ([§20.1.9](#)) or the `notifyAll` method ([§20.1.10](#)) for this object, or a certain amount of real time has elapsed.

This method may be called only when the current thread is already synchronized on this object. If the current thread does not own the lock on this object, an `IllegalMonitorStateException` is thrown.

This method causes the current thread (call it *T*) to place itself in the wait set ([§17.14](#)) for this object and then to relinquish any and all synchronization claims on this object. Thread *T* becomes disabled for thread scheduling purposes and lies dormant until one of four things happens:

- Some other thread invokes the `notify` method for this object and thread *T* happens to be arbitrarily chosen as the thread to be awakened.
- Some other thread invokes the `notifyAll` method for this object.
- Some other thread interrupts ([§20.20.31](#)) thread *T*.
- The specified amount of real time has elapsed, more or less. The amount of real time, measured in milliseconds, is given by `millis`. If `millis` is zero, however, then real time is not taken into consideration and the thread simply waits until notified.

The thread *T* is then removed from the wait set for this object and re-enabled for thread scheduling. It then competes in the usual manner with other threads for the right to synchronize on the object; once it has gained control of the object, all its synchronization claims on the object are restored to the status quo ante—that is, to the situation as of the time that the `wait` method was invoked. Thread *T* then returns from the invocation of the `wait` method. Thus, on return from the `wait` method, the synchronization state of the object and of thread *T* is exactly as it was when the `wait` method was invoked.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then an `InterruptedException` is thrown. This exception is not thrown until the lock status of this object has been restored as described above.

Note that the `wait` method, as it places the current thread into the wait set for this object, unlocks only this object; any other objects on which the current thread may be synchronized remain locked while the thread waits.

20.1.8 `public final void wait(long millis, int nanos)`
 throws `IllegalMonitorStateException`, `InterruptedException`

This method causes the current thread to wait until either some other thread invokes the `notify` method ([§20.1.9](#)) or the `notifyAll` method ([§20.1.10](#)) for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

The amount of real time, measured in nanoseconds, is given by:

`1000000*millis+nanos`

In all other respects, this method does the same thing as the method `wait` of one argument ([§20.1.7](#)). In particular, `wait(0, 0)` means the same thing as `wait(0)`.

20.1.9 `public final void notify()`
throws `IllegalMonitorStateException`

If any threads are waiting ([§20.1.7](#)) on this object, one of them is chosen to be awakened. The choice is arbitrary and at the discretion of the implementation.

The `notify` method may be called only when the current thread is already synchronized on this object. If the current thread does not own the lock on this object, an `IllegalMonitorStateException` is thrown.

The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object. The awakened thread will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened thread enjoys no reliable privilege or disadvantage in being the next thread to lock this object.

20.1.10 `public final void notifyAll()`
throws `IllegalMonitorStateException`

All the threads waiting ([§20.1.7](#)) on this object are awakened.

The `notifyAll` method may be called only when the current thread is already synchronized on this object. If the current thread does not own the lock on this object, an `IllegalMonitorStateException` is thrown.

The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object. The awakened threads will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened threads enjoy no reliable privilege or disadvantage in being the next thread to lock this object.

20.1.11 `protected void finalize() throws Throwable`

The general contract of `finalize` is that it is invoked if and when the Java Virtual Machine has determined that there is no longer any means by which this object can be accessed by any thread that has not yet died ([§12.7](#)), except as a result of an action taken by the finalization of some other object or class which is ready to be finalized. The `finalize` method may take any action, including making this object available again to other threads; the usual purpose of `finalize`, however, is to perform cleanup actions before the object is irrevocably discarded. For example, the `finalize` method for an object that represents an input/output connection might perform explicit I/O transactions to break the connection before the object is permanently discarded.

The `finalize` method of class `Object` performs no special action; it simply returns normally. Subclasses of `Object` may override this definition.

Java does not guarantee which thread will invoke the `finalize` method for any given object. It is guaranteed, however, that the thread that invokes `finalize` will not be holding any user-visible synchronization locks when `finalize` is invoked. If an uncaught exception is thrown by the `finalize` method, the exception is ignored and finalization of that object terminates.

After the `finalize` method has been invoked for an object, no further action is taken until the Java Virtual Machine has again determined that there is no longer any means by which this object can be accessed by any thread that has not yet died, including possible actions by other objects or classes which are ready to be finalized, at which point the object may be discarded.

The `finalize` method is never invoked more than once by a Java Virtual Machine for any given object.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.2 The Interface `java.lang.Cloneable`

The `Cloneable` interface should be implemented by any class that is intended to support or override the method `clone` ([§20.1.5](#)).

```
public interface Cloneable { }
```

The interface `Cloneable` declares no methods.

I am disappointed in Japp. He has no method!

--Agatha Christie, *The Mysterious Affair at Styles* (1920), Chapter 8

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.3 The Class java.lang.Class

Instances of the class `Class` represent classes and interfaces in a way that can be manipulated by a running Java program. Every array also belongs to a class represented by a `Class` object that is shared among all arrays with the same element type and number of dimensions.

There is no public constructor for the class `Class`. The Java Virtual Machine automatically constructs `Class` objects as classes are loaded; such objects cannot be created by user programs.

```
public final class Class {
    public String toString();
    public String getName();
    public boolean isInterface();
    public Class getSuperclass();
    public Class[] getInterfaces();
    public Object newInstance()
        throws InstantiationException, IllegalAccessException;
    public static Class forName(String className)
        throws ClassNotFoundException;
    public ClassLoader getClassLoader();
}
```

20.3.1 public String toString()

If this `Class` object represents a class (which may be a declared class or an array class), a string is returned consisting of the word `class`, a space, and the name of the class as returned by the `getName` method ([§20.3.2](#)). If this `Class` object represents an interface, a string is returned consisting of the word `interface`, a space, and the name of the interface as returned by the `getName` method.

In other words, this method returns a string equal to the value of:

```
(isInterface() ? "interface " : "class ") + getName()
```

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.3.2 public String getName()

The fully qualified name of the class or interface represented by this `Class` object is returned as a `String`. For example:

```
new Object().getClass().getName()
```

returns `"java.lang.Object"`.

If this class object represents a class of arrays, then the name consists of the name of the element type in Java signature format, preceded by one or more `"["` characters representing the depth of array nesting. For example:


```
(new Object[3]).getClass().getName()
```

returns "[Ljava.lang.Object;" and:

```
(new int[3][4][5][6][7][8][9]).getClass().getName()
```

returns "[[[[[[I". The encoding of element type names is as follows:

B	byte	
C	char	
D	double	
F	float	
I	int	
J	long	
Lclassname;		class or interface
S	short	
Z	boolean	

A class or interface name *classname* is given in fully qualified form as shown in the example above. For a full description of type descriptors see the chapter on the format of class files in the *Java Virtual Machine Specification*.

20.3.3 `public boolean isInterface()`

If this `Class` object represents an interface, `true` is returned. If this `Class` object represents a class, `false` is returned.

20.3.4 `public Class getSuperclass()`

If this `Class` object represents any class other than the class `Object`, then the `Class` that represents the superclass of that class is returned. If this `Class` object is the one that represents the class `Object`, or if it represents an interface, `null` is returned. If this `Class` object represents an array class, then the `Class` that represents class `Object` is returned.

20.3.5 `public Class[] getInterfaces()`

This method returns an array of objects that represent interfaces. The array may be empty.

If this `Class` object represents a class, the array contains objects representing all interfaces directly implemented by the class. The order of the interface objects in the array corresponds to the order of the interface names in the `implements` clause of the declaration of the class represented by this `Class` object. For example, given the class declaration:

```
class Shimmer implements FloorWax, DessertTopping { ... }
```

suppose the value of `s` is an instance of `Shimmer`; the value of the expression:


```
s.getClass().getInterfaces()[0]
```

is the `Class` object that represents interface `FloorWax`; and the value of:

```
s.getClass().getInterfaces()[1]
```

is the `Class` object that represents interface `DessertTopping`.

If this `Class` object represents an interface, the array contains objects representing all interfaces directly extended by the interface—that is, the immediate superinterfaces of the interface. The order of the interface objects in the array corresponds to the order of the interface names in the `extends` clause of the declaration of the interface represented by this `Class` object.

20.3.6 `public Object newInstance()`
throws `InstantiationException`, `IllegalAccessException`

This method creates and returns a new instance of the class represented by this `Class` object. This is done exactly as if by a class instance creation expression ([§15.8](#)) with an empty argument list; for example, if `t` is the `Class` object that represents class `Thread`, then `_t.newInstance()` does exactly the same thing as `new Thread()`. If evaluation of such a class instance creation expression would complete abruptly, then the call to the `newInstance` method will complete abruptly for the same reason. See also [§11.5.1.2](#) for more on `InstantiationException`.

20.3.7 `public ClassLoader getClassLoader()`

This method returns a reference to the class loader ([§20.14](#)) that loaded this class. If this class has no class loader, then `null` is returned.

20.3.8 `public static Class forName(String className)`
throws `ClassNotFoundException`

Given the fully-qualified name of a class as a string, this method attempts to locate, load, and link the class ([§12.2](#)). If it succeeds, then a reference to the `Class` object for the class is returned. If it fails, then a `ClassNotFoundException` is thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.4 The Class java.lang.Boolean

Objects of type `Boolean` represent primitive values of type `boolean`.

```
public final class Boolean {
    public static final Boolean TRUE = new Boolean(true);
    public static final Boolean FALSE = new Boolean(false);
    public Boolean(boolean value);
    public Boolean(String s);
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public boolean booleanValue();
    public static Boolean valueOf(String s);
    public static boolean getBoolean(String name);
}
```

20.4.1 `public static final Boolean TRUE = new Boolean(true);`

The constant value of this field is a `Boolean` object corresponding to the primitive value `true`.

20.4.2 `public static final Boolean FALSE = new Boolean(false);`

The constant value of this field is a `Boolean` object corresponding to the primitive value `false`.

20.4.3 `public Boolean(boolean value)`

This constructor initializes a newly created `Boolean` object so that it represents the primitive value that is the argument.

20.4.4 `public Boolean(String s)`

This constructor initializes a newly created `Boolean` object so that it represents `true` if and only if the argument is not `null` and is equal, ignoring case, to the string `"true"`.

Examples:

```
new Boolean("True") produces a Boolean object that represents true.
new Boolean("yes") produces a Boolean object that represents false.
```

20.4.5 `public boolean booleanValue()`

The primitive `boolean` value represented by this `Boolean` object is returned.

20.4.6 `public String toString()`

If this `Boolean` object represents `true`, a string equal to `"true"` is returned. If this `Boolean` object represents `false`, a string equal to `"false"` is returned.

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.4.7 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Boolean` object that represents the same boolean value as this `Boolean` object.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

20.4.8 `public int hashCode()`

If this `Boolean` object represents `true`, the integer 1231 is returned. If this `Boolean` object represents `false`, the integer 1237 is returned.

Overrides the `hashCode` method of `Object` ([§20.1.4](#)).

20.4.9 `public static boolean valueOf(String s)`

The result is `true` if and only if the argument is not `null` and is equal, ignoring case, to the string `"true"`.

Example: `Boolean.valueOf("True")` returns `true`.

Example: `Boolean.valueOf("yes")` returns `false`.

20.4.10 `public static boolean getBoolean(String name)`

The result is `true` if and only if the value of the system property (§20.18.9) named by the argument is equal, ignoring case, to the string `"true"`.

*This above all: to thine own self be true,
And it must follow, as the night the day,
Thou canst not then be false to any man.
--William Shakespeare, Hamlet, Act I, scene iii*

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.5 The Class java.lang.Character

Here is the whole set! a character dead at every word.

--Richard Brinsley Sheridan, The School for Scandal, Act 2, scene 2

Objects of type `Character` represent primitive values of type `char`.

```
public final class Character {
    public static final char MIN_VALUE = '\u0000';
    public static final char MAX_VALUE = '\uffff';
    public static final int MIN_RADIX = 2;
    public static final int MAX_RADIX = 36;
    public Character(char value);
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public char charValue();
    public static boolean isDefined(char ch);
    public static boolean isLowerCase(char ch);
    public static boolean isUpperCase(char ch);
    public static boolean isTitleCase(char ch);
    public static boolean isDigit(char ch);
    public static boolean isLetter(char ch);
    public static boolean isLetterOrDigit(char ch);
    public static boolean isJavaLetter(char ch);
    public static boolean isJavaLetterOrDigit(char ch);
    public static boolean isSpace(char ch);
    public static char toLowerCase(char ch);
    public static char toUpperCase(char ch);
    public static char toTitleCase(char ch);
    public static int digit(char ch, int radix);
    public static char forDigit(int digit, int radix);
}
```

Many of the methods of class `Character` are defined in terms of a "Unicode attribute table" that specifies a name for every defined Unicode character as well as other possible attributes, such as a decimal value, an uppercase equivalent, a lowercase equivalent, and/or a titlecase equivalent. Prior to Java 1.1, these methods were internal to the Java compiler and based on Unicode 1.1.5, as described here. The most recent versions of these methods should be used in Java compilers that are to run on Java systems that do not yet include these methods.

The Unicode 1.1.5 attribute table is available on the World Wide Web as:

<ftp://unicode.org/pub/MappingTables/UnicodeData-1.1.5.txt>

However, this file contains a few errors. The term "Unicode attribute table" in the following sections refers to the contents of this file after the following corrections have been applied:

- The following entries should have titlecase mappings as shown here:
03D0;GREEK BETA SYMBOL;Ll;0;L;;;;;N;GREEK SMALL LETTER CURLED BETA;;0392;;0392

03D1;GREEK THETA SYMBOL;Li;0;L;;;;;N;GREEK SMALL LETTER SCRIPT THETA;;0398;;0398

03D5;GREEK PHI SYMBOL;Li;0;L;;;;;N;GREEK SMALL LETTER SCRIPT PHI;;03A6;;03A6

03D6;GREEK PI SYMBOL;Li;0;L;;;;;N;GREEK SMALL LETTER OMEGA PI;;03A0;;03A0

03F0;GREEK KAPPA SYMBOL;Li;0;L;;;;;N;GREEK SMALL LETTER SCRIPT KAPPA;;039A;;039A

03F1;GREEK RHO SYMBOL;Li;0;L;;;;;N;GREEK SMALL LETTER TAILED RHO;;03A1;;03A1

- The following entries should have numeric values as shown here:

FF10;FULLWIDTH DIGIT ZERO;Nd;0;EN;0030;0;0;0;N;;;;;

FF11;FULLWIDTH DIGIT ONE;Nd;0;EN;0031;1;1;1;N;;;;;

FF12;FULLWIDTH DIGIT TWO;Nd;0;EN;0032;2;2;2;N;;;;;

FF13;FULLWIDTH DIGIT THREE;Nd;0;EN;0033;3;3;3;N;;;;;

FF14;FULLWIDTH DIGIT FOUR;Nd;0;EN;0034;4;4;4;N;;;;;

FF15;FULLWIDTH DIGIT FIVE;Nd;0;EN;0035;5;5;5;N;;;;;

FF16;FULLWIDTH DIGIT SIX;Nd;0;EN;0036;6;6;6;N;;;;;

FF17;FULLWIDTH DIGIT SEVEN;Nd;0;EN;0037;7;7;7;N;;;;;

FF18;FULLWIDTH DIGIT EIGHT;Nd;0;EN;0038;8;8;8;N;;;;;

FF19;FULLWIDTH DIGIT NINE;Nd;0;EN;0039;9;9;9;N;;;;;

- The following entries should have no lowercase equivalents:

03DA;GREEK LETTER STIGMA;Lu;0;L;;;;;N;GREEK CAPITAL LETTER STIGMA;;;

03DC;GREEK LETTER DIGAMMA;Lu;0;L;;;;;N;GREEK CAPITAL LETTER DIGAMMA;;;

03DE;GREEK LETTER KOPPA;Lu;0;L;;;;;N;GREEK CAPITAL LETTER KOPPA;;;

03E0;GREEK LETTER SAMPI;Lu;0;L;;;;;N;GREEK CAPITAL LETTER SAMPI;;;

- This entry should have uppercase and titlecase equivalents as shown here:

03C2;GREEK SMALL LETTER FINAL SIGMA;Li;0;L;;;;;N;;;03A3;;03A3

It is anticipated that these problems will be corrected for Unicode version 2.0.

Java 1.1 will include the methods defined here, either based on Unicode 1.1.5 or, we hope, updated versions of the methods that use the newer Unicode 2.0. The character attribute table for Unicode 2.0 is currently available on the World Wide Web as the file:

<ftp://unicode.org/pub/MappingTables/UnicodeData-2.0.12.txt>

If you are implementing a Java compiler or system, please refer to the page:

<http://java.sun.com/Series>

which will be updated with information about the Unicode-dependent methods.

The biggest change in Unicode 2.0 is a complete rearrangement of the Korean Hangul characters. There are numerous smaller improvements as well.

It is our intention that Java will track Unicode as it evolves over time. Given that full Unicode support is just emerging in the marketplace, and that changes in Unicode are in areas which are not yet widely used, this should cause minimal problems and further Java's goal of worldwide language support.

20.5.1 `public static final char MIN_VALUE = '\u0000';`

The constant value of this field is the smallest value of type `char`.

[This field is scheduled for introduction in Java version 1.1.]

20.5.2 `public static final char MAX_VALUE = '\uffff';`

The constant value of this field is the smallest value of type `char`.

[This field is scheduled for introduction in Java version 1.1.]

20.5.3 `public static final int MIN_RADIX = 2;`

The constant value of this field is the smallest value permitted for the radix argument in radix-conversion methods such as the `digit` method ([§20.5.23](#)), the `forDigit` method ([§20.5.24](#)), and the `toString` method of class `Integer` ([§20.7](#)).

20.5.4 `public static final int MAX_RADIX = 36;`

The constant value of this field is the largest value permitted for the radix argument in radix-conversion methods such as the `digit` method ([§20.5.23](#)), the `forDigit` method ([§20.5.24](#)), and the `toString` method of class `Integer` ([§20.7](#)).

20.5.5 `public Character(char value)`

This constructor initializes a newly created `Character` object so that it represents the primitive value that is the argument.

20.5.6 `public String toString()`

The result is a `String` whose length is 1 and whose sole component is the primitive `char` value represented by this `Character` object.

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.5.7 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Character` object that represents the same `char` value as this `Character` object.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

20.5.8 `public int hashCode()`

The result is the primitive `char` value represented by this `Character` object, cast to type `int`.

Overrides the `hashCode` method of `Object` ([§20.1.4](#)).

20.5.9 `public char charValue()`

The primitive `char` value represented by this `Character` object is returned.

20.5.10 `public static boolean isDefined(char ch)`

The result is `true` if and only if the character argument is a defined Unicode character.

A character is a defined Unicode character if and only if at least one of the following is true:

- It has an entry in the Unicode attribute table.
- It is not less than `\u3040` and not greater than `\u9FA5`.
- It is not less than `\uF900` and not greater than `\uFA2D`.

It follows, then, that for Unicode 1.1.5 as corrected above, the defined Unicode characters are exactly those with codes in the following list, which contains both single codes and inclusive ranges:

0000-01F5, 01FA-0217, 0250-02A8, 02B0-02DE, 02E0-02E9, 0300-0345, 0360-0361, 0374-0375, 037A, 037E, 0384-038A, 038C, 038E-03A1, 03A3-03CE, 03D0-03D6, 03DA, 03DC, 03DE, 03E0, 03E2-03F3, 0401-040C, 040E-044F, 0451-045C, 045E-0486, 0490-04C4, 04C7-04C8, 04CB-04CC, 04D0-04EB, 04EE-04F5, 04F8-04F9, 0531-0556, 0559-055F, 0561-0587, 0589, 05B0-05B9, 05BB-05C3, 05D0-05EA, 05F0-05F4, 060C, 061B, 061F, 0621-063A, 0640-0652, 0660-066D, 0670-06B7, 06BA-06BE, 06C0-06CE, 06D0-06ED, 06F0-06F9, 0901-0903, 0905-0939, 093C-094D, 0950-0954, 0958-0970, 0981-0983, 0985-098C, 098F-0990, 0993-09A8, 09AA-09B0, 09B2, 09B6-09B9, 09BC, 09BE-09C4, 09C7-09C8, 09CB-09CD, 09D7, 09DC-09DD, 09DF-09E3, 09E6-09FA, 0A02, 0A05-0A0A, 0A0F-0A10, 0A13-0A28, 0A2A-0A30, 0A32-0A33, 0A35-0A36, 0A38-0A39, 0A3C, 0A3E-0A42, 0A47-0A48, 0A4B-0A4D, 0A59-0A5C, 0A5E, 0A66-0A74, 0A81-0A83, 0A85-0A8B, 0A8D, 0A8F-0A91, 0A93-0AA8, 0AAA-0AB0, 0AB2-0AB3, 0AB5-0AB9, 0ABC-0AC5, 0AC7-0AC9, 0ACB-0ACD, 0AD0, 0AE0, 0AE6-0AEF, 0B01-0B03, 0B05-0B0C, 0B0F-0B10, 0B13-0B28, 0B2A-0B30, 0B32-0B33, 0B36-0B39, 0B3C-0B43, 0B47-0B48, 0B4B-0B4D, 0B56-0B57, 0B5C-0B5D, 0B5F-0B61, 0B66-0B70, 0B82-0B83, 0B85-

0B8A, 0B8E-0B90, 0B92-0B95, 0B99-0B9A, 0B9C, 0B9E-0B9F, 0BA3-0BA4, 0BA8-0BAA, 0BAE-0BB5, 0BB7-0BB9, 0BBE-0BC2, 0BC6-0BC8, 0BCA-0BCD, 0BD7, 0BE7-0BF2, 0C01-0C03, 0C05-0C0C, 0C0E-0C10, 0C12-0C28, 0C2A-0C33, 0C35-0C39, 0C3E-0C44, 0C46-0C48, 0C4A-0C4D, 0C55-0C56, 0C60-0C61, 0C66-0C6F, 0C82-0C83, 0C85-0C8C, 0C8E-0C90, 0C92-0CA8, 0CAA-0CB3, 0CB5-0CB9, 0CBE-0CC4, 0CC6-0CC8, 0CCA-0CCD, 0CD5-0CD6, 0CDE, 0CE0-0CE1, 0CE6-0CEF, 0D02-0D03, 0D05-0D0C, 0D0E-0D10, 0D12-0D28, 0D2A-0D39, 0D3E-0D43, 0D46-0D48, 0D4A-0D4D, 0D57, 0D60-0D61, 0D66-0D6F, 0E01-0E3A, 0E3F-0E5B, 0E81-0E82, 0E84, 0E87-0E88, 0E8A, 0E8D, 0E94-0E97, 0E99-0E9F, 0EA1-0EA3, 0EA5, 0EA7, 0EAA-0EAB, 0EAD-0EB9, 0EBB-0EBD, 0EC0-0EC4, 0EC6, 0EC8-0ECD, 0ED0-0ED9, 0EDC-0EDD, 10A0-10C5, 10D0-10F6, 10FB, 1100-1159, 115F-11A2, 11A8-11F9, 1E00-1E9A, 1EA0-1EF9, 1F00-1F15, 1F18-1F1D, 1F20-1F45, 1F48-1F4D, 1F50-1F57, 1F59, 1F5B, 1F5D, 1F5F-1F7D, 1F80-1FB4, 1FB6-1FC4, 1FC6-1FD3, 1FD6-1FDB, 1FDD-1FEF, 1FF2-1FF4, 1FF6-1FFE, 2000-202E, 2030-2046, 206A-2070, 2074-208E, 20A0-20AA, 20D0-20E1, 2100-2138, 2153-2182, 2190-21EA, 2200-22F1, 2300, 2302-237A, 2400-2424, 2440-244A, 2460-24EA, 2500-2595, 25A0-25EF, 2600-2613, 261A-266F, 2701-2704, 2706-2709, 270C-2727, 2729-274B, 274D, 274F-2752, 2756, 2758-275E, 2761-2767, 2776-2794, 2798-27AF, 27B1-27BE, 3000-3037, 303F, 3041-3094, 3099-309E, 30A1-30FE, 3105-312C, 3131-318E, 3190-319F, 3200-321C, 3220-3243, 3260-327B, 327F-32B0, 32C0-32CB, 32D0-32FE, 3300-3376, 337B-33DD, 33E0-33FE, 3400-9FA5, F900-FA2D, FB00-FB06, FB13-FB17, FB1E-FB36, FB38-FB3C, FB3E, FB40-FB41, FB43-FB44, FB46-FBB1, FBD3-FD3F, FD50-FD8F, FD92-FDC7, FDF0-FDFB, FE20-FE23, FE30-FE44, FE49-FE52, FE54-FE66, FE68-FE6B, FE70-FE72, FE74, FE76-FEFC, FEFF, FF01-FF5E, FF61-FFBE, FFC2-FFC7, FFCA-FFCF, FFD2-FFD7, FFDA-FFDC, FFE0-FFE6, FFE8-FFEE, FFFD.

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.11 `public static boolean isLowerCase(char ch)`

The result is `true` if and only if the character argument is a lowercase character.

A character is considered to be lowercase if and only if all of the following are true:

- The character `ch` is not in the range `\u2000` through `\u2FFF`.
- The Unicode attribute table does not specify a mapping to lowercase for this character (the purpose of this requirement is to exclude titlecase characters).
- At least one of the following is true:
 - The Unicode attribute table specifies a mapping to uppercase for this character.
 - The name for the character in the Unicode attribute table contains the words `SMALL LETTER` or the words `SMALL LIGATURE`.

It follows, then, that for Unicode 1.1.5 as corrected above, the lowercase Unicode characters are exactly those with codes in the following list, which contains both single codes and inclusive ranges: 0061-007A, 00DF-00F6, 00F8-00FF, 0101-0137 (odds only), 0138-0148 (evens only), 0149-0177 (odds only), 017A-017E (evens only), 017F-0180, 0183, 0185, 0188, 018C-018D, 0192, 0195, 0199-019B, 019E, 01A1-01A5 (odds only), 01A8, 01AB, 01AD, 01B0, 01B4, 01B6, 01B9-01BA, 01BD, 01C6, 01C9, 01CC-01DC (evens only), 01DD-01EF (odds only), 01F0, 01F3, 01F5, 01FB-0217 (odds only), 0250-0261, 0263-0269, 026B-0273, 0275, 0277-027F, 0282-028E, 0290-0293, 029A, 029D-029E, 02A0, 02A3-02A8, 0390, 03AC-03CE, 03D0-03D1, 03D5-03D6, 03E3-03EF (odds only), 03F0-03F1, 0430-044F, 0451-045C, 045E-045F, 0461-0481 (odds only),

0491-04BF (odds only), 04C2, 04C4, 04C8, 04CC, 04D1-04EB (odds only), 04EF-04F5 (odds only), 04F9, 0561-0587, 1E01-1E95 (odds only), 1E96-1E9A, 1EA1-1EF9 (odds only), 1F00-1F07, 1F10-1F15, 1F20-1F27, 1F30-1F37, 1F40-1F45, 1F50-1F57, 1F60-1F67, 1F70-1F7D, 1F80-1F87, 1F90-1F97, 1FA0-1FA7, 1FB0-1FB4, 1FB6-1FB7, 1FC2-1FC4, 1FC6-1FC7, 1FD0-1FD3, 1FD6-1FD7, 1FE0-1FE7, 1FF2-1FF4, 1FF6-1FF7, FB00-FB06, FB13-FB17, FF41-FF5A.

Of the first 128 Unicode characters, exactly 26 are considered to be lowercase:

abcdefghijklmnopqrstuvwxyz

[This specification for the method `isLowerCase` is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns `false` for all arguments larger than `\u00FF`.]

20.5.12 `public static boolean isUpperCase(char ch)`

The result is `true` if and only if the character argument is an uppercase character.

A character is considered to be uppercase if and only if all of the following are true:

- The character `ch` is not in the range `\u2000` through `\u2FFF`.
- The Unicode attribute table does not specify a mapping to uppercase for this character (the purpose of this requirement is to exclude titlecase characters).
- At least one of the following is true:
 - The Unicode attribute table specifies a mapping to lowercase for this character.
 - The name for the character in the Unicode attribute table contains the words `CAPITAL LETTER` or the words `CAPITAL LIGATURE`.

It follows, then, that for Unicode 1.1.5 as corrected above, the uppercase Unicode characters are exactly those with codes in the following list, which contains both single codes and inclusive ranges: 0041-005A, 00C0-00D6, 00D8-00DE, 0100-0136 (evens only), 0139-0147 (odds only), 014A-0178 (evens only), 0179-017D (odds only), 0181-0182, 0184, 0186, 0187, 0189-018B, 018E-0191, 0193-0194, 0196-0198, 019C-019D, 019F-01A0, 01A2, 01A4, 01A7, 01A9, 01AC, 01AE, 01AF, 01B1-01B3, 01B5, 01B7, 01B8, 01BC, 01C4, 01C7, 01CA, 01CD-01DB (odds only), 01DE-01EE (evens only), 01F1, 01F4, 01FA-0216 (evens only), 0386, 0388-038A, 038C, 038E, 038F, 0391-03A1, 03A3-03AB, 03E2-03EE (evens only), 0401-040C, 040E-042F, 0460-0480 (evens only), 0490-04BE (evens only), 04C1, 04C3, 04C7, 04CB, 04D0-04EA (evens only), 04EE-04F4 (evens only), 04F8, 0531-0556, 10A0-10C5, 1E00-1E94 (evens only), 1EA0-1EF8 (evens only), 1F08-1F0F, 1F18-1F1D, 1F28-1F2F, 1F38-1F3F, 1F48-1F4D, 1F59-1F5F (odds only), 1F68-1F6F, 1F88-1F8F, 1F98-1F9F, 1FA8-1FAF, 1FB8-1FBC, 1FC8-1FCC, 1FD8-1FDB, 1FE8-1FEC, 1FF8-1FFC, FF21-FF3A.

Of the first 128 Unicode characters, exactly 26 are considered to be uppercase:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

[This specification for the method `isUpperCase` is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns `false` for all arguments larger than `\u00FF`.]

20.5.13 `public static boolean isTitleCase(char ch)`

The result is `true` if and only if the character argument is a titlecase character.

The notion of "titlecase" was introduced into Unicode to handle a peculiar situation: there are single Unicode characters whose appearance in each case looks exactly like two ordinary Latin letters. For example, there is a single Unicode character ``LJ'` (`\u01C7`) that looks just like the characters ``L'` and ``J'` put together. There is a corresponding lowercase letter ``lj'` (`\u01C9`) as well. These characters are present in Unicode primarily to allow one-to-one translations from the Cyrillic alphabet, as used in Serbia, for example, to the Latin alphabet. Now suppose the word "LJUBINJE" (which has *six* characters, not eight, because two of them are the single Unicode characters ``LJ'` and ``NJ'`, perhaps produced by one-to-one translation from the Cyrillic) is to be written as part of a book title, in capitals and lowercase. The strategy of making the first letter uppercase and the rest lowercase results in "LJubinke"-most unfortunate. The solution is that there must be a third form, called a *titlecase* form. The titlecase form of ``LJ'` is ``Lj'` (`\u01C8`) and the titlecase form of ``NJ'` is ``Nj'`. A word for a book title is then best rendered by converting the first letter to titlecase if possible, otherwise to uppercase; the remaining letters are then converted to lowercase.

A character is considered to be titlecase if and only if both of the following are true:

- The character `ch` is not in the range `\u2000` through `\u2FFF`.
- The Unicode attribute table specifies a mapping to uppercase *and* a mapping to lowercase for this character.

There are exactly four Unicode 1.1.5 characters for which `isTitleCase` returns `true`:

<code>\u01C5</code>	LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
<code>\u01C8</code>	LATIN CAPITAL LETTER L WITH SMALL LETTER J
<code>\u01CB</code>	LATIN CAPITAL LETTER N WITH SMALL LETTER J
<code>\u01F2</code>	LATIN CAPITAL LETTER D WITH SMALL LETTER Z

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.14 `public static boolean isDigit(char ch)`

The result is `true` if and only if the character argument is a digit.

A character is considered to be a digit if and only if both of the following are true:

- The character `ch` is not in the range `\u2000` through `\u2FFF`.
- The name for the character in the Unicode attribute table contains the word `DIGIT`.

The digits are those characters with the following codes:

0030-0039	ISO-Latin-1 (and ASCII) digits ('0'-'9')
0660-0669	Arabic-Indic digits
06F0-06F9	Eastern Arabic-Indic digits
0966-096F	Devanagari digits
09E6-09EF	Bengali digits

0A66-0A6F	Gurmukhi digits
0AE6-0AEF	Gujarati digits
0B66-0B6F	Oriya digits
0BE7-0BEF	Tamil digits (there are only nine of these-no zero digit)
0C66-0C6F	Telugu digits
0CE6-0CEF	Kannada digits
0D66-0D6F	Malayalam digits
0E50-0E59	Thai digits
0ED0-0ED9	Lao digits
FF10-FF19	Fullwidth digits

Of the first 128 Unicode characters, exactly 10 are considered to be digits:

0123456789

[This specification for the method `isDigit` is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns `false` for all arguments larger than `\u00FF`.]

20.5.15 `public static boolean isLetter(char ch)`

The result is `true` if and only if the character argument is a letter.

A character is considered to be a letter if and only if it is a letter or digit ([§20.5.16](#)) but is not a digit ([§20.5.14](#)).

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.16 `public static boolean isLetterOrDigit(char ch)`

The result is `true` if and only if the character argument is a letter-or-digit.

A character is considered to be a letter-or-digit if and only if it is a defined Unicode character ([§20.5.10](#)) and its code lies in one of the following ranges:

0030-0039	ISO-Latin-1 (and ASCII) digits ('0'-'9')
0041-005A	ISO-Latin-1 (and ASCII) uppercase Latin letters ('A'-'Z')
0061-007A	ISO-Latin-1 (and ASCII) lowercase Latin letters ('a'-'z')
00C0-00D6	ISO-Latin-1 supplementary letters
00D8-00F6	ISO-Latin-1 supplementary letters
00F8-00FF	ISO-Latin-1 supplementary letters
0100-1FFF	Latin extended-A, Latin extended-B, IPA extensions, spacing modifier letters, combining diacritical marks, basic Greek, Greek symbols and Coptic, Cyrillic, Armenian, Hebrew extended-A, Basic Hebrew, Hebrew extended-B, Basic Arabic, Arabic extended, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Basic Georgian, Georgian extended, Hanguljamo, Latin extended additional, Greek extended
3040-9FFF	Hiragana, Katakana, Bopomofo, Hangul compatibility Jamo, CJK miscellaneous, enclosed CJK characters and months, CJK compatibility, Hangul, Hangul

supplementary-A, Hangul supplementary-B, CJK unified
 ideographs
 F900-FDFF CJK compatibility ideographs, alphabetic presentation
 forms, Arabic presentation forms-A
 FE70-FEFE Arabic presentation forms-B
 FF10-FF19 Fullwidth digits
 FF21-FF3A Fullwidth Latin uppercase
 FF41-FF5A Fullwidth Latin lowercase
 FF66-FFDC Halfwidth Katakana and Hangul

It follows, then, that for Unicode 1.1.5 as corrected above, the Unicode letters and digits are exactly those with codes in the following list, which contains both single codes and inclusive ranges:

0030-0039, 0041-005A, 0061-007A, 00C0-00D6, 00D8-00F6, 00F8-01F5, 01FA-0217, 0250-02A8,
 02B0-02DE, 02E0-02E9, 0300-0345, 0360-0361, 0374-0375, 037A, 037E, 0384-038A, 038C, 038E,
 038F-03A1, 03A3-03CE, 03D0-03D6, 03DA-03E2, 03DA, 03DC, 03DE, 03E0, 03E2-03F3, 0401-
 040C, 040E-044F, 0451-045C, 045E-0486, 0490-04C4, 04C7-04C8, 04CB-04CC, 04D0-04EB,
 04EE-04F5, 04F8-04F9, 0531-0556, 0559-055F, 0561-0587, 0589, 05B0-05B9, 05BB-05C3, 05D0-
 05EA, 05F0-05F4, 060C, 061B, 061F, 0621, 0622-063A, 0640-0652, 0660-066D, 0670-06B7,
 06BA-06BE, 06C0-06CE, 06D0-06ED, 06F0-06F9, 0901-0903, 0905-0939, 093C-094D, 0950-0954,
 0958-0970, 0981-0983, 0985-098C, 098F-0990, 0993-09A8, 09AA-09B0, 09B2, 09B6-09B9, 09BC,
 09BE, 09BF-09C4, 09C7-09C8, 09CB-09CD, 09D7, 09DC-09DD, 09DF-09E3, 09E6-09FA, 0A02,
 0A05-0A0A, 0A0F-0A10, 0A13-0A28, 0A2A-0A30, 0A32-0A33, 0A35-0A36, 0A38-0A39, 0A3C,
 0A3E, 0A3F-0A42, 0A47-0A48, 0A4B-0A4D, 0A59-0A5C, 0A5E, 0A66-0A74, 0A81-0A83, 0A85-
 0A8B, 0A8D, 0A8F, 0A90-0A91, 0A93-0AA8, 0AAA-0AB0, 0AB2-0AB3, 0AB5-0AB9, 0ABC-0AC5,
 0AC7-0AC9, 0ACB-0ACD, 0AD0, 0AE0, 0AE6-0AEF, 0B01-0B03, 0B05-0B0C, 0B0F-0B10, 0B13-
 0B28, 0B2A-0B30, 0B32-0B33, 0B36-0B39, 0B3C-0B43, 0B47-0B48, 0B4B-0B4D, 0B56-0B57,
 0B5C-0B5D, 0B5F-0B61, 0B66-0B70, 0B82-0B83, 0B85-0B8A, 0B8E-0B90, 0B92-0B95, 0B99-
 0B9A, 0B9C, 0B9E, 0B9F, 0BA3-0BA4, 0BA8-0BAA, 0BAE-0BB5, 0BB7-0BB9, 0BBE-0BC2, 0BC6-
 0BC8, 0BCA-0BCD, 0BD7, 0BE7-0BF2, 0C01-0C03, 0C05-0C0C, 0C0E-0C10, 0C12-0C28,
 0C2A-0C33, 0C35-0C39, 0C3E-0C44, 0C46-0C48, 0C4A-0C4D, 0C55-0C56, 0C60-0C61, 0C66-
 0C6F, 0C82-0C83, 0C85-0C8C, 0C8E-0C90, 0C92-0CA8, 0CAA-0CB3, 0CB5-0CB9, 0CBE-0CC4,
 0CC6-0CC8, 0CCA-0CCD, 0CD5-0CD6, 0CDE, 0CE0, 0CE1, 0CE6-0CEF, 0D02-0D03, 0D05-0D0C,
 0D0E-0D10, 0D12-0D28, 0D2A-0D39, 0D3E-0D43, 0D46-0D48, 0D4A-0D4D, 0D57, 0D60-0D61,
 0D66-0D6F, 0E01-0E3A, 0E3F-0E5B, 0E81-0E82, 0E84, 0E87-0E88, 0E8A, 0E8D, 0E94-0E97,
 0E99-0E9F, 0EA1-0EA3, 0EA5, 0EA7, 0EAA-0EAB, 0EAD-0EB9, 0EBB-0EBD, 0EC0-0EC4, 0EC6,
 0EC8, 0EC9-0ECD, 0ED0-0ED9, 0EDC-0EDD, 10A0-10C5, 10D0-10F6, 10FB, 1100-1159,
 115F-11A2, 11A8-11F9, 1E00-1E9A, 1EA0-1EF9, 1F00-1F15, 1F18-1F1D, 1F20-1F45, 1F48-1F4D,
 1F50-1F57, 1F59, 1F5B, 1F5D, 1F5F-1F7D, 1F80-1FB4, 1FB6-1FC4, 1FC6-1FD3, 1FD6-1FDB,
 1FDD-1FEF, 1FF2-1FF4, 1FF6-1FFE, 3041-3094, 3099-309E, 30A1-30FE, 3105-312C, 3131-318E,
 3190-319F, 3200-321C, 3220-3243, 3260-327B, 327F-32B0, 32C0-32CB, 32D0-32FE, 3300-3376,
 337B-33DD, 33E0-33FE, 3400-9FA5, F900-FA2D, FB00-FB06, FB13-FB17, FB1E-FB36, FB38-
 FB3C, FB3E, FB40, FB41, FB43, FB44, FB46, FB47-FBB1, FBD3-FD3F, FD50-FD8F, FD92-FDC7,
 FDF0-FDFB, FE70-FE72, FE74, FE76, FE77-FEFC, FF10-FF19, FF21-FF3A, FF41-FF5A, FF66-
 FFBE, FFC2-FFC7, FFCA-FFCF, FFD2-FFD7, FFDA-FFDC.

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.17 `public static boolean isJavaLetter(char ch)`

The result is true if and only if the character argument is a character that can begin a Java identifier.

A character is considered to be a Java letter if and only if it is a letter ([§20.5.15](#)) or is the dollar sign character '\$' (\u0024) or the underscore ("low line") character '_' (\u005F).

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.18 `public static boolean isJavaLetterOrDigit(char ch)`

The result is `true` if and only if the character argument is a character that can occur in a Java identifier after the first character.

A character is considered to be a Java letter-or-digit if and only if it is a letter-or-digit ([§20.5.16](#)) or is the dollar sign character '\$' (\u0024) or the underscore ("low line") character '_' (\u005F).

[This method is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#).]

20.5.19 `public static boolean isSpace(char ch)`

The result is `true` if the argument `ch` is one of the following characters:

'\t'	\u0009	HT	HORIZONTAL TABULATION
'\n'	\u000A	LF	LINE FEED (also known as NEW LINE)
'\f'	\u000C	FF	FORM FEED
'\r'	\u000D	CR	CARRIAGE RETURN
' '	\u0020	SP	SPACE

Otherwise, the result is `false`.

20.5.20 `public static char toLowerCase(char ch)`

If the character `ch` has a lowercase equivalent specified in the Unicode attribute table, then that lowercase equivalent character is returned. Otherwise, the argument `ch` is returned.

The lowercase equivalents specified in the Unicode attribute table, for Unicode 1.1.5 as corrected above, are as follows, where character codes to the right of arrows are the lowercase equivalents of character codes to the left of arrows: 0041-005A{ewc msdncd, EWGraphic, LNG5x 0 /a "langref.BMP"}0061-007A, 00C0-00D6{ewc msdncd, EWGraphic, LNG5x 1 /a "langref.BMP"}00E0-00F6, 00D8-00DE{ewc msdncd, EWGraphic, LNG5x 2 /a "langref.BMP"}00F8-00FE, 0100-012E{ewc msdncd, EWGraphic, LNG5x 3 /a "langref.BMP"}0101-012F (evens to odds), 0132-0136{ewc msdncd, EWGraphic, LNG5x 4 /a "langref.BMP"}0133-0137 (evens to odds), 0139-0147{ewc msdncd, EWGraphic, LNG5x 5 /a "langref.BMP"}013A-0148 (odds to evens), 014A-0176{ewc msdncd, EWGraphic, LNG5x 6 /a "langref.BMP"}014B-0177 (evens to odds), 0178{ewc msdncd, EWGraphic, LNG5x 7 /a "langref.BMP"}00FF, 0179-017D{ewc msdncd, EWGraphic, LNG5x 8 /a "langref.BMP"}017A-017E (odds to evens), 0181{ewc msdncd, EWGraphic, LNG5x 9 /a "langref.BMP"}0253, 0182{ewc msdncd, EWGraphic, LNG5x 10 /a "langref.BMP"}0183, 0184{ewc msdncd, EWGraphic, LNG5x 11 /a "langref.BMP"}0185, 0186{ewc msdncd, EWGraphic, LNG5x 12 /a "langref.BMP"}0254, 0187{ewc

msdncd, EWGraphic, LNG5x 13 /a "langref.BMP"}0188, 018A{ewc msdncd, EWGraphic, LNG5x 14 /a "langref.BMP"}0257, 018B{ewc msdncd, EWGraphic, LNG5x 15 /a "langref.BMP"}018C, 018E{ewc msdncd, EWGraphic, LNG5x 16 /a "langref.BMP"}0258, 018F{ewc msdncd, EWGraphic, LNG5x 17 /a "langref.BMP"}0259, 0190{ewc msdncd, EWGraphic, LNG5x 18 /a "langref.BMP"}025B, 0191{ewc msdncd, EWGraphic, LNG5x 19 /a "langref.BMP"}0192, 0193{ewc msdncd, EWGraphic, LNG5x 20 /a "langref.BMP"}0260, 0194{ewc msdncd, EWGraphic, LNG5x 21 /a "langref.BMP"}0263, 0196{ewc msdncd, EWGraphic, LNG5x 22 /a "langref.BMP"}0269, 0197{ewc msdncd, EWGraphic, LNG5x 23 /a "langref.BMP"}0268, 0198{ewc msdncd, EWGraphic, LNG5x 24 /a "langref.BMP"}0199, 019C{ewc msdncd, EWGraphic, LNG5x 25 /a "langref.BMP"}026F, 019D{ewc msdncd, EWGraphic, LNG5x 26 /a "langref.BMP"}0272, 01A0-01A4{ewc msdncd, EWGraphic, LNG5x 27 /a "langref.BMP"}01A1-01A5 (evens to odds), 01A7{ewc msdncd, EWGraphic, LNG5x 28 /a "langref.BMP"}01A8, 01A9{ewc msdncd, EWGraphic, LNG5x 29 /a "langref.BMP"}0283, 01AC{ewc msdncd, EWGraphic, LNG5x 30 /a "langref.BMP"}01AD, 01AE{ewc msdncd, EWGraphic, LNG5x 31 /a "langref.BMP"}0288, 01AF{ewc msdncd, EWGraphic, LNG5x 32 /a "langref.BMP"}01B0, 01B1{ewc msdncd, EWGraphic, LNG5x 33 /a "langref.BMP"}028A, 01B2{ewc msdncd, EWGraphic, LNG5x 34 /a "langref.BMP"}028B, 01B3{ewc msdncd, EWGraphic, LNG5x 35 /a "langref.BMP"}01B4, 01B5{ewc msdncd, EWGraphic, LNG5x 36 /a "langref.BMP"}01B6, 01B7{ewc msdncd, EWGraphic, LNG5x 37 /a "langref.BMP"}0292, 01B8{ewc msdncd, EWGraphic, LNG5x 38 /a "langref.BMP"}01B9, 01BC{ewc msdncd, EWGraphic, LNG5x 39 /a "langref.BMP"}01BD, 01C4{ewc msdncd, EWGraphic, LNG5x 40 /a "langref.BMP"}01C6, 01C5{ewc msdncd, EWGraphic, LNG5x 41 /a "langref.BMP"}01C6, 01C7{ewc msdncd, EWGraphic, LNG5x 42 /a "langref.BMP"}01C9, 01C8{ewc msdncd, EWGraphic, LNG5x 43 /a "langref.BMP"}01C9, 01CA{ewc msdncd, EWGraphic, LNG5x 44 /a "langref.BMP"}01CC, 01CB-01DB{ewc msdncd, EWGraphic, LNG5x 45 /a "langref.BMP"}01CC-01DC (odds to evens), 01DE-01EE{ewc msdncd, EWGraphic, LNG5x 46 /a "langref.BMP"}01DF-01EF (evens to odds), 01F1{ewc msdncd, EWGraphic, LNG5x 47 /a "langref.BMP"}01F3, 01F2{ewc msdncd, EWGraphic, LNG5x 48 /a "langref.BMP"}01F3, 01F4{ewc msdncd, EWGraphic, LNG5x 49 /a "langref.BMP"}01F5, 01FA-0216{ewc msdncd, EWGraphic, LNG5x 50 /a "langref.BMP"}01FB-0217 (evens to odds), 0386{ewc msdncd, EWGraphic, LNG5x 51 /a "langref.BMP"}03AC, 0388-038A{ewc msdncd, EWGraphic, LNG5x 52 /a "langref.BMP"}03AD-03AF, 038C{ewc msdncd, EWGraphic, LNG5x 53 /a "langref.BMP"}03CC, 038E{ewc msdncd, EWGraphic, LNG5x 54 /a "langref.BMP"}03CD, 038F{ewc msdncd, EWGraphic, LNG5x 55 /a "langref.BMP"}03CE, 0391-03A1{ewc msdncd, EWGraphic, LNG5x 56 /a "langref.BMP"}03B1-03C1, 03A3-03AB{ewc msdncd, EWGraphic, LNG5x 57 /a "langref.BMP"}03C3-03CB, 03E2-03EE{ewc msdncd, EWGraphic, LNG5x 58 /a "langref.BMP"}03E3-03EF (evens to odds), 0401-040C{ewc msdncd, EWGraphic, LNG5x 59 /a "langref.BMP"}0451-045C, 040E{ewc msdncd, EWGraphic, LNG5x 60 /a "langref.BMP"}045E, 040F{ewc msdncd, EWGraphic, LNG5x 61 /a "langref.BMP"}045F, 0410-042F{ewc msdncd, EWGraphic, LNG5x 62 /a "langref.BMP"}0430-044F, 0460-0480{ewc msdncd, EWGraphic, LNG5x 63 /a "langref.BMP"}0461-0481 (evens to odds), 0490-04BE{ewc msdncd, EWGraphic, LNG5x 64 /a "langref.BMP"}0491-04BF (evens to odds), 04C1{ewc msdncd, EWGraphic,

LNG5x 65 /a "langref.BMP"}04C2, 04C3{ewc msdncd, EWGraphic, LNG5x 66 /a "langref.BMP"}04C4, 04C7{ewc msdncd, EWGraphic, LNG5x 67 /a "langref.BMP"}04C8, 04CB{ewc msdncd, EWGraphic, LNG5x 68 /a "langref.BMP"}04CC, 04D0-04EA{ewc msdncd, EWGraphic, LNG5x 69 /a "langref.BMP"}04D1-04EB (evens to odds), 04EE-04F4{ewc msdncd, EWGraphic, LNG5x 70 /a "langref.BMP"}04EF-04F5 (evens to odds), 04F8{ewc msdncd, EWGraphic, LNG5x 71 /a "langref.BMP"}04F9, 0531-0556{ewc msdncd, EWGraphic, LNG5x 72 /a "langref.BMP"}0561-0586, 10A0-10C5{ewc msdncd, EWGraphic, LNG5x 73 /a "langref.BMP"}10D0-10F5, 1E00-1E94{ewc msdncd, EWGraphic, LNG5x 74 /a "langref.BMP"}1E01-1E95 (evens to odds), 1EA0-1EF8{ewc msdncd, EWGraphic, LNG5x 75 /a "langref.BMP"}1EA1-1EF9 (evens to odds), 1F08-1F0F{ewc msdncd, EWGraphic, LNG5x 76 /a "langref.BMP"}1F00-1F07, 1F18-1F1D{ewc msdncd, EWGraphic, LNG5x 77 /a "langref.BMP"}1F10-1F15, 1F28-1F2F{ewc msdncd, EWGraphic, LNG5x 78 /a "langref.BMP"}1F20-1F27, 1F38-1F3F{ewc msdncd, EWGraphic, LNG5x 79 /a "langref.BMP"}1F30-1F37, 1F48-1F4D{ewc msdncd, EWGraphic, LNG5x 80 /a "langref.BMP"}1F40-1F45, 1F59{ewc msdncd, EWGraphic, LNG5x 81 /a "langref.BMP"}1F51, 1F5B{ewc msdncd, EWGraphic, LNG5x 82 /a "langref.BMP"}1F53, 1F5D{ewc msdncd, EWGraphic, LNG5x 83 /a "langref.BMP"}1F55, 1F5F{ewc msdncd, EWGraphic, LNG5x 84 /a "langref.BMP"}1F57, 1F68-1F6F{ewc msdncd, EWGraphic, LNG5x 85 /a "langref.BMP"}1F60-1F67, 1F88-1F8F{ewc msdncd, EWGraphic, LNG5x 86 /a "langref.BMP"}1F80-1F87, 1F98-1F9F{ewc msdncd, EWGraphic, LNG5x 87 /a "langref.BMP"}1F90-1F97, 1FA8-1FAF{ewc msdncd, EWGraphic, LNG5x 88 /a "langref.BMP"}1FA0-1FA7, 1FB8{ewc msdncd, EWGraphic, LNG5x 89 /a "langref.BMP"}1FB0, 1FB9{ewc msdncd, EWGraphic, LNG5x 90 /a "langref.BMP"}1FB1, 1FBA{ewc msdncd, EWGraphic, LNG5x 91 /a "langref.BMP"}1F70, 1FBB{ewc msdncd, EWGraphic, LNG5x 92 /a "langref.BMP"}1F71, 1FBC{ewc msdncd, EWGraphic, LNG5x 93 /a "langref.BMP"}1FB3, 1FC8-1FCB{ewc msdncd, EWGraphic, LNG5x 94 /a "langref.BMP"}1F72-1F75, 1FCC{ewc msdncd, EWGraphic, LNG5x 95 /a "langref.BMP"}1FC3, 1FD8{ewc msdncd, EWGraphic, LNG5x 96 /a "langref.BMP"}1FD0, 1FD9{ewc msdncd, EWGraphic, LNG5x 97 /a "langref.BMP"}1FD1, 1FDA{ewc msdncd, EWGraphic, LNG5x 98 /a "langref.BMP"}1F76, 1FDB{ewc msdncd, EWGraphic, LNG5x 99 /a "langref.BMP"}1F77, 1FE8{ewc msdncd, EWGraphic, LNG5x 100 /a "langref.BMP"}1FE0, 1FE9{ewc msdncd, EWGraphic, LNG5x 101 /a "langref.BMP"}1FE1, 1FEA{ewc msdncd, EWGraphic, LNG5x 102 /a "langref.BMP"}1F7A, 1FEB{ewc msdncd, EWGraphic, LNG5x 103 /a "langref.BMP"}1F7B, 1FEC{ewc msdncd, EWGraphic, LNG5x 104 /a "langref.BMP"}1FE5, 1FF8{ewc msdncd, EWGraphic, LNG5x 105 /a "langref.BMP"}1F78, 1FF9{ewc msdncd, EWGraphic, LNG5x 106 /a "langref.BMP"}1F79, 1FFA{ewc msdncd, EWGraphic, LNG5x 107 /a "langref.BMP"}1F7C, 1FFB{ewc msdncd, EWGraphic, LNG5x 108 /a "langref.BMP"}1F7D, 1FFC{ewc msdncd, EWGraphic, LNG5x 109 /a "langref.BMP"}1FF3, 2160-216F{ewc msdncd, EWGraphic, LNG5x 110 /a "langref.BMP"}2170-217F, 24B6-24CF{ewc msdncd, EWGraphic, LNG5x 111 /a "langref.BMP"}24D0-24E9, FF21-FF3A{ewc msdncd, EWGraphic, LNG5x 112 /a "langref.BMP"}FF41-FF5A.

Note that the method `isLowerCase` ([§20.5.11](#)) will not necessarily return `true` when given the result of the `toLowerCase` method.

[This specification for the method `toLowerCase` is scheduled for introduction in Java version 1.1,

either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns its argument for all arguments larger than `\u00FF`.]

20.5.21 `public static char toUpperCase(char ch)`

If the character `ch` has an uppercase equivalent specified in the Unicode attribute table, then that uppercase equivalent character is returned. Otherwise, the argument `ch` is returned.

The uppercase equivalents specified in the Unicode attribute table for Unicode 1.1.5 as corrected above, are as follows, where character codes to the right of arrows are the uppercase equivalents of character codes to the left of arrows:

0061-007A{ewc msdncd, EWGraphic, LNG5x 113 /a "langref.BMP"}0041-005A, 00E0-00F6{ewc msdncd, EWGraphic, LNG5x 114 /a "langref.BMP"}00C0-00D6, 00F8-00FE{ewc msdncd, EWGraphic, LNG5x 115 /a "langref.BMP"}00D8-00DE, 00FF{ewc msdncd, EWGraphic, LNG5x 116 /a "langref.BMP"}0178, 0101-012F{ewc msdncd, EWGraphic, LNG5x 117 /a "langref.BMP"}0100-012E (odds to evens), 0133-0137{ewc msdncd, EWGraphic, LNG5x 118 /a "langref.BMP"}0132-0136 (odds to evens), 013A-0148{ewc msdncd, EWGraphic, LNG5x 119 /a "langref.BMP"}0139-0147 (evens to odds), 014B-0177{ewc msdncd, EWGraphic, LNG5x 120 /a "langref.BMP"}014A-0176 (odds to evens), 017A-017E{ewc msdncd, EWGraphic, LNG5x 121 /a "langref.BMP"}0179-017D (evens to odds), 017F{ewc msdncd, EWGraphic, LNG5x 122 /a "langref.BMP"}0053, 0183-0185{ewc msdncd, EWGraphic, LNG5x 123 /a "langref.BMP"}0182-0184 (odds to evens), 0188{ewc msdncd, EWGraphic, LNG5x 124 /a "langref.BMP"}0187, 018C{ewc msdncd, EWGraphic, LNG5x 125 /a "langref.BMP"}018B, 0192{ewc msdncd, EWGraphic, LNG5x 126 /a "langref.BMP"}0191, 0199{ewc msdncd, EWGraphic, LNG5x 127 /a "langref.BMP"}0198, 01A1-01A5{ewc msdncd, EWGraphic, LNG5x 128 /a "langref.BMP"}01A0-01A4 (odds to evens), 01A8{ewc msdncd, EWGraphic, LNG5x 129 /a "langref.BMP"}01A7, 01AD{ewc msdncd, EWGraphic, LNG5x 130 /a "langref.BMP"}01AC, 01B0{ewc msdncd, EWGraphic, LNG5x 131 /a "langref.BMP"}01AF, 01B4{ewc msdncd, EWGraphic, LNG5x 132 /a "langref.BMP"}01B3, 01B6{ewc msdncd, EWGraphic, LNG5x 133 /a "langref.BMP"}01B5, 01B9{ewc msdncd, EWGraphic, LNG5x 134 /a "langref.BMP"}01B8, 01BD{ewc msdncd, EWGraphic, LNG5x 135 /a "langref.BMP"}01BC, 01C5{ewc msdncd, EWGraphic, LNG5x 136 /a "langref.BMP"}01C4, 01C6{ewc msdncd, EWGraphic, LNG5x 137 /a "langref.BMP"}01C4, 01C8{ewc msdncd, EWGraphic, LNG5x 138 /a "langref.BMP"}01C7, 01C9{ewc msdncd, EWGraphic, LNG5x 139 /a "langref.BMP"}01C7, 01CB{ewc msdncd, EWGraphic, LNG5x 140 /a "langref.BMP"}01CA, 01CC{ewc msdncd, EWGraphic, LNG5x 141 /a "langref.BMP"}01CA, 01CE-01DC{ewc msdncd, EWGraphic, LNG5x 142 /a "langref.BMP"}01CD-01DB (evens to odds), 01DF-01EF{ewc msdncd, EWGraphic, LNG5x 143 /a "langref.BMP"}01DE-01EE (odds to evens), 01F2{ewc msdncd, EWGraphic, LNG5x 144 /a "langref.BMP"}01F1, 01F3{ewc msdncd, EWGraphic, LNG5x 145 /a "langref.BMP"}01F1, 01F5{ewc msdncd, EWGraphic, LNG5x 146 /a "langref.BMP"}01F4, 01FB-0217{ewc msdncd, EWGraphic, LNG5x 147 /a "langref.BMP"}01FA-0216 (odds to evens), 0253{ewc msdncd, EWGraphic, LNG5x 148 /a "langref.BMP"}0181, 0254{ewc msdncd, EWGraphic, LNG5x 149 /a "langref.BMP"}0186, 0257{ewc msdncd, EWGraphic, LNG5x 150 /a "langref.BMP"}018A, 0258{ewc msdncd, EWGraphic, LNG5x 151 /a "langref.BMP"}018E, 0259{ewc msdncd, EWGraphic, LNG5x 152 /a "langref.BMP"}018F, 025B{ewc msdncd,

EWGraphic, LNG5x 153 /a "langref.BMP"}0190, 0260{ewc msdncd, EWGraphic, LNG5x 154 /a "langref.BMP"}0193, 0263{ewc msdncd, EWGraphic, LNG5x 155 /a "langref.BMP"}0194, 0268{ewc msdncd, EWGraphic, LNG5x 156 /a "langref.BMP"}0197, 0269{ewc msdncd, EWGraphic, LNG5x 157 /a "langref.BMP"}0196, 026F{ewc msdncd, EWGraphic, LNG5x 158 /a "langref.BMP"}019C, 0272{ewc msdncd, EWGraphic, LNG5x 159 /a "langref.BMP"}019D, 0283{ewc msdncd, EWGraphic, LNG5x 160 /a "langref.BMP"}01A9, 0288{ewc msdncd, EWGraphic, LNG5x 161 /a "langref.BMP"}01AE, 028A{ewc msdncd, EWGraphic, LNG5x 162 /a "langref.BMP"}01B1, 028B{ewc msdncd, EWGraphic, LNG5x 163 /a "langref.BMP"}01B2, 0292{ewc msdncd, EWGraphic, LNG5x 164 /a "langref.BMP"}01B7, 03AC{ewc msdncd, EWGraphic, LNG5x 165 /a "langref.BMP"}0386, 03AD-03AF{ewc msdncd, EWGraphic, LNG5x 166 /a "langref.BMP"}0388-038A, 03B1-03C1{ewc msdncd, EWGraphic, LNG5x 167 /a "langref.BMP"}0391-03A1, 03C2{ewc msdncd, EWGraphic, LNG5x 168 /a "langref.BMP"}03A3, 03C3-03CB{ewc msdncd, EWGraphic, LNG5x 169 /a "langref.BMP"}03A3-03AB, 03CC{ewc msdncd, EWGraphic, LNG5x 170 /a "langref.BMP"}038C, 03CD{ewc msdncd, EWGraphic, LNG5x 171 /a "langref.BMP"}038E, 03CE{ewc msdncd, EWGraphic, LNG5x 172 /a "langref.BMP"}038F, 03D0{ewc msdncd, EWGraphic, LNG5x 173 /a "langref.BMP"}0392, 03D1{ewc msdncd, EWGraphic, LNG5x 174 /a "langref.BMP"}0398, 03D5{ewc msdncd, EWGraphic, LNG5x 175 /a "langref.BMP"}03A6, 03D6{ewc msdncd, EWGraphic, LNG5x 176 /a "langref.BMP"}03A0, 03E3-03EF{ewc msdncd, EWGraphic, LNG5x 177 /a "langref.BMP"}03E2-03EE (odds to evens), 03F0{ewc msdncd, EWGraphic, LNG5x 178 /a "langref.BMP"}039A, 03F1{ewc msdncd, EWGraphic, LNG5x 179 /a "langref.BMP"}03A1, 0430-044F{ewc msdncd, EWGraphic, LNG5x 180 /a "langref.BMP"}0410-042F, 0451-045C{ewc msdncd, EWGraphic, LNG5x 181 /a "langref.BMP"}0401-040C, 045E{ewc msdncd, EWGraphic, LNG5x 182 /a "langref.BMP"}040E, 045F{ewc msdncd, EWGraphic, LNG5x 183 /a "langref.BMP"}040F, 0461-0481{ewc msdncd, EWGraphic, LNG5x 184 /a "langref.BMP"}0460-0480 (odds to evens), 0491-04BF{ewc msdncd, EWGraphic, LNG5x 185 /a "langref.BMP"}0490-04BE (odds to evens), 04C2{ewc msdncd, EWGraphic, LNG5x 186 /a "langref.BMP"}04C1, 04C4{ewc msdncd, EWGraphic, LNG5x 187 /a "langref.BMP"}04C3, 04C8{ewc msdncd, EWGraphic, LNG5x 188 /a "langref.BMP"}04C7, 04CC{ewc msdncd, EWGraphic, LNG5x 189 /a "langref.BMP"}04CB, 04D1-04EB{ewc msdncd, EWGraphic, LNG5x 190 /a "langref.BMP"}04D0-04EA (odds to evens), 04EF-04F5{ewc msdncd, EWGraphic, LNG5x 191 /a "langref.BMP"}04EE-04F4 (odds to evens), 04F9{ewc msdncd, EWGraphic, LNG5x 192 /a "langref.BMP"}04F8, 0561-0586{ewc msdncd, EWGraphic, LNG5x 193 /a "langref.BMP"}0531-0556, 1E01-1E95{ewc msdncd, EWGraphic, LNG5x 194 /a "langref.BMP"}1E00-1E94 (odds to evens), 1EA1-1EF9{ewc msdncd, EWGraphic, LNG5x 195 /a "langref.BMP"}1EA0-1EF8 (odds to evens), 1F00-1F07{ewc msdncd, EWGraphic, LNG5x 196 /a "langref.BMP"}1F08-1F0F, 1F10-1F15{ewc msdncd, EWGraphic, LNG5x 197 /a "langref.BMP"}1F18-1F1D, 1F20-1F27{ewc msdncd, EWGraphic, LNG5x 198 /a "langref.BMP"}1F28-1F2F, 1F30-1F37{ewc msdncd, EWGraphic, LNG5x 199 /a "langref.BMP"}1F38-1F3F, 1F40-1F45{ewc msdncd, EWGraphic, LNG5x 200 /a "langref.BMP"}1F48-1F4D, 1F51{ewc msdncd, EWGraphic, LNG5x 201 /a "langref.BMP"}1F59, 1F53{ewc msdncd, EWGraphic, LNG5x 202 /a "langref.BMP"}1F5B, 1F55{ewc msdncd, EWGraphic,

LNG5x 203 /a "langref.BMP"}1F5D, 1F57{ewc msdncd, EWGraphic, LNG5x 204 /a "langref.BMP"}1F5F, 1F60-1F67{ewc msdncd, EWGraphic, LNG5x 205 /a "langref.BMP"}1F68-1F6F, 1F70{ewc msdncd, EWGraphic, LNG5x 206 /a "langref.BMP"}1FBA, 1F71{ewc msdncd, EWGraphic, LNG5x 207 /a "langref.BMP"}1FBB, 1F72-1F75{ewc msdncd, EWGraphic, LNG5x 208 /a "langref.BMP"}1FC8-1FCB, 1F76{ewc msdncd, EWGraphic, LNG5x 209 /a "langref.BMP"}1FDA, 1F77{ewc msdncd, EWGraphic, LNG5x 210 /a "langref.BMP"}1FDB, 1F78{ewc msdncd, EWGraphic, LNG5x 211 /a "langref.BMP"}1FF8, 1F79{ewc msdncd, EWGraphic, LNG5x 212 /a "langref.BMP"}1FF9, 1F7A{ewc msdncd, EWGraphic, LNG5x 213 /a "langref.BMP"}1FEA, 1F7B{ewc msdncd, EWGraphic, LNG5x 214 /a "langref.BMP"}1FEB, 1F7C{ewc msdncd, EWGraphic, LNG5x 215 /a "langref.BMP"}1FFA, 1F7D{ewc msdncd, EWGraphic, LNG5x 216 /a "langref.BMP"}1FFB, 1F80-1F87{ewc msdncd, EWGraphic, LNG5x 217 /a "langref.BMP"}1F88-1F8F, 1F90-1F97{ewc msdncd, EWGraphic, LNG5x 218 /a "langref.BMP"}1F98-1F9F, 1FA0-1FA7{ewc msdncd, EWGraphic, LNG5x 219 /a "langref.BMP"}1FA8-1FAF, 1FB0{ewc msdncd, EWGraphic, LNG5x 220 /a "langref.BMP"}1FB8, 1FB1{ewc msdncd, EWGraphic, LNG5x 221 /a "langref.BMP"}1FB9, 1FB3{ewc msdncd, EWGraphic, LNG5x 222 /a "langref.BMP"}1FBC, 1FC3{ewc msdncd, EWGraphic, LNG5x 223 /a "langref.BMP"}1FCC, 1FD0{ewc msdncd, EWGraphic, LNG5x 224 /a "langref.BMP"}1FD8, 1FD1{ewc msdncd, EWGraphic, LNG5x 225 /a "langref.BMP"}1FD9, 1FE0{ewc msdncd, EWGraphic, LNG5x 226 /a "langref.BMP"}1FE8, 1FE1{ewc msdncd, EWGraphic, LNG5x 227 /a "langref.BMP"}1FE9, 1FE5{ewc msdncd, EWGraphic, LNG5x 228 /a "langref.BMP"}1FEC, 1FF3{ewc msdncd, EWGraphic, LNG5x 229 /a "langref.BMP"}1FFC, 2170-217F{ewc msdncd, EWGraphic, LNG5x 230 /a "langref.BMP"}2160-216F, 24D0-24E9{ewc msdncd, EWGraphic, LNG5x 231 /a "langref.BMP"}24B6-24CF, FF41-FF5A{ewc msdncd, EWGraphic, LNG5x 232 /a "langref.BMP"}FF21-FF3A.

Note that the method `isUpperCase` ([§20.5.12](#)) will not necessarily return `true` when given the result of the `toUpperCase` method.

[This specification for the method `toUpperCase` is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns its argument for all arguments larger than `\u00FE`. Note that although `\u00FF` is a lowercase character, its uppercase equivalent is `\u0178`; `toUpperCase` in versions of Java prior to version 1.1 simply do not consistently handle or use Unicode character codes above `\u00FF`.]

20.5.22 `public static char toTitleCase(char ch)`

If the character `ch` has a titlecase equivalent specified in the Unicode attribute table, then that titlecase equivalent character is returned; otherwise, the argument `ch` is returned.

Note that the method `isTitleCase` ([§20.5.13](#)) will not necessarily return `true` when given the result of the `toTitleCase` method. The Unicode attribute table always has the titlecase attribute equal to the uppercase attribute for characters that have uppercase equivalents but no separate titlecase form.

Example: `Character.toTitleCase('a')` returns `'A'`

Example: `Character.toTitleCase('Q')` returns `'Q'`

Example: `Character.toTitleCase('lj')` returns `'Lj'` where `'lj'` is the Unicode character `\u01C9` and `'Lj'` is its titlecase equivalent character `\u01C8`.

[This method is scheduled for introduction in Java version 1.1.]

20.5.23 `public static int digit(char ch, int radix)`

Returns the numeric value of the character `ch` considered as a digit in the specified radix. If the value of `radix` is not a valid radix, or the character `ch` is not a valid digit in the specified radix, then `-1` is returned.

A radix is valid if and only if its value is not less than `Character.MIN_RADIX` ([§20.5.3](#)) and not greater than `Character.MAX_RADIX` ([§20.5.4](#)).

A character is a valid digit if and only if one of the following is true:

- The method `isDigit` returns `true` for the character, and the decimal digit value of the character, as specified in the Unicode attribute table, is less than the specified radix. In this case, the decimal digit value is returned.
- The character is one of the uppercase Latin letters `'A'-'Z'` (`\u0041-\u005A`) and its code is less than `radix+'A'-10`. In this case `ch-'A'+10` is returned.
- The character is one of the lowercase Latin letters `'a'-'z'` (`\u0061-\u007A`) and its code is less than `radix+'a'-10`. In this case `ch-'a'+10` is returned.

[This specification for the method `digit` is scheduled for introduction in Java version 1.1, either as defined here, or updated for Unicode 2.0; see [§20.5](#). In previous versions of Java, this method returns `-1` for all character codes larger than `\u00FF`.]

20.5.24 `public static char forDigit(int digit, int radix)`

Returns a character that represents the given digit in the specified radix. If the value of `radix` is not a valid radix, or the value of `digit` is not a valid digit in the specified radix, the null character `'\u0000'` is returned.

A radix is valid if and only if its value is not less than `Character.MIN_RADIX` ([§20.5.3](#)) and not greater than `Character.MAX_RADIX` ([§20.5.4](#)).

A digit is valid if and only if it is nonnegative and less than the `radix`.

If the digit is less than 10, then the character value `'0'+digit` is returned; otherwise, `'a'+digit-10` is returned. Thus, the digits produced by `forDigit`, in increasing order of value, are the ASCII characters:

0123456789abcdefghijklmnopqrstuvwxyz

(these are `'\u0030'` through `'\u0039'` and `'\u0061'` through `'\u007a'`). If uppercase letters are desired, the `toUpperCase` method may be called on the result:

`Character.toUpperCase(Character.forDigit(digit, radix))`

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.6 The Class java.lang.Number

The abstract class `Number` has subclasses `Integer`, `Long`, `Float`, and `Double` which wrap primitive types, defining abstract methods to convert the represented numeric value to `int`, `long`, `float`, and `double`.

```
public abstract class Number {
    public abstract int intValue();
    public abstract long longValue();
    public abstract float floatValue();
    public abstract double doubleValue();
}
```

20.6.1 public abstract int intValue()

The general contract of the `intValue` method is that it returns the numeric value represented by this `Number` object after converting it to type `int`.

Overridden by `Integer` ([§20.7.8](#)), `Long` ([§20.8.8](#)), `Float` ([§20.9.12](#)), and `Double` ([§20.10.11](#)).

20.6.2 public abstract long longValue()

The general contract of the `longValue` method is that it returns the numeric value represented by this `Number` object after converting it to type `long`.

Overridden by `Integer` ([§20.7.9](#)), `Long` ([§20.8.9](#)), `Float` ([§20.9.13](#)), and `Double` ([§20.10.12](#)).

20.6.3 public abstract float floatValue()

The general contract of the `floatValue` method is that it returns the numeric value represented by this `Number` object after converting it to type `float`.

Overridden by `Integer` ([§20.7.10](#)), `Long` ([§20.8.10](#)), `Float` ([§20.9.14](#)), and `Double` ([§20.10.13](#)).

20.6.4 public abstract double doubleValue()

The general contract of the `doubleValue` method is that it returns the numeric value represented by this `Number` object after converting it to type `double`.

Overridden by `Integer` ([§20.7.11](#)), `Long` ([§20.8.11](#)), `Float` ([§20.9.15](#)), and `Double` ([§20.10.14](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.7 The Class java.lang.Integer

```
public final class Integer extends Number {
    public static final int MIN_VALUE = 0x80000000;
    public static final int MAX_VALUE = 0x7fffffff;
    public Integer(int value);
    public Integer(String s)

        throws NumberFormatException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public int intValue();
    public long longValue();
    public float floatValue();
    public double doubleValue();
    public static String toString(int i);
    public static String toString(int i, int radix);
    public static String toHexString(long i);
    public static String toOctalString(long i);
    public static String toBinaryString(long i);
    public static int parseInt(String s)

        throws NumberFormatException;
    public static int parseInt(String s, int radix)

        throws NumberFormatException;
    public static Integer valueOf(String s)

        throws NumberFormatException;
    public static Integer valueOf(String s, int radix)

        throws NumberFormatException;
    public static Integer getInteger(String nm);
    public static Integer getInteger(String nm, int val);
    public static Integer getInteger(String nm, Integer val);
}
```

20.7.1 `public static final int MIN_VALUE = 0x80000000;`

The constant value of this field is -2147483648, the lowest value of type int.

20.7.2 `public static final int MAX_VALUE = 0x7fffffff;`

The constant value of this field is 2147483647, the highest value of type int.

20.7.3 `public Integer(int value)`

This constructor initializes a newly created `Integer` object so that it represents the primitive value that is the argument.

20.7.4 `public Integer(String s) throws NumberFormatException`

This constructor initializes a newly created `Integer` object so that it represents the integer represented by the string in decimal form. The string is converted to an `int` in exactly the manner used by the `parseInt` method ([§20.7.18](#)) for radix 10.

20.7.5 `public String toString()`

The integer value represented by this `Integer` object is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString` method that takes one argument ([§20.7.12](#)).

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.7.6 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is an `Integer` object that represents the same `int` value as this `Integer` object.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

20.7.7 `public int hashCode()`

The result is the primitive `int` value represented by this `Integer` object.

Overrides the `hashCode` method of `Object` ([§20.1.4](#)).

20.7.8 `public int intValue()`

The `int` value represented by this `Integer` object is returned.

Overrides the `intValue` method of `Number` ([§20.6.1](#)).

20.7.9 `public long longValue()`

The `int` value represented by this `Integer` object is converted ([§5.1.2](#)) to type `long` and the result of the conversion is returned.

Overrides the `longValue` method of `Number` ([§20.6.2](#)).

20.7.10 `public float floatValue()`

The `int` value represented by this `Integer` object is converted ([§5.1.2](#)) to type `float` and the result of the conversion is returned.

Overrides the `floatValue` method of `Number` ([§20.6.3](#)).

20.7.11 `public double doubleValue()`

The `int` value represented by this `Integer` object is converted ([§5.1.2](#)) to type `double` and the result of the conversion is returned.

Overrides the `doubleValue` method of `Number` [\(§20.6.4\)](#).

20.7.12 `public static String toString(int i)`

The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and the radix 10 were given as arguments to the `toString` method that takes two arguments [\(§20.7.13\)](#).

20.7.13 `public static String toString(int i, int radix)`

The first argument is converted to a signed representation in the radix specified by the second argument; this representation is returned as a string.

If the `radix` is smaller than `Character.MIN_RADIX` [\(§20.5.3\)](#) or larger than `Character.MAX_RADIX` [\(§20.5.4\)](#), then the value 10 is used instead.

If the first argument is negative, the first character of the result will be the character '-' ('`\u002d`'). If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character '0' ('`\u0030`'); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

0123456789abcdefghijklmnopqrstuvwxyz

These are '`\u0030`' through '`\u0039`' and '`\u0061`' through '`\u007a`'. If the `radix` is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are 0123456789abcdef. If uppercase letters are desired, the `toUpperCase` method [\(§20.12.36\)](#) of class `String` may be called on the result:

`Integer.toString(n, 16).toUpperCase()`

20.7.14 `public static String toHexString(int i)`

The argument is converted to an unsigned representation in hexadecimal radix (base 16); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {ewc msdn cd, EWGraphic, LNG7x 0 /a "langref.BMP"} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character '0' ('`\u0030`'); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as hexadecimal digits:

0123456789abcdef

These are the characters '`\u0030`' through '`\u0039`' and '`\u0061`' through '`\u0066`'. If uppercase letters are desired, the `toUpperCase` method [\(§20.12.36\)](#) of class `String` may be

called on the result:

```
Long.toHexString(n).toUpperCase()
```

20.7.15 `public static String toOctalString(int i)`

The argument is converted to an unsigned representation in octal radix (base 8); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {[EWC msdn](#)cd, EWGraphic, LNG7x 1 /a "langref.BMP"} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character '0' ('[\u0030](#)'); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The octal digits are:

```
01234567
```

These are the characters '[\u0030](#)' through '[\u0037](#)'.

20.7.16 `public static String toBinaryString(int i)`

The argument is converted to an unsigned representation in binary radix (base 2); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {[EWC msdn](#)cd, EWGraphic, LNG7x 2 /a "langref.BMP"} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character '0' ('[\u0030](#)'); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters '0' ('[\u0030](#)') and '1' ('[\u0031](#)') are used as binary digits.

20.7.17 `public static int parseInt(String s)`
 throws `NumberFormatException`

The argument is interpreted as representing a signed decimal integer. The components of the string must all be decimal digits, except that the first character may be '-' ('[\u002d](#)') to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt` method that takes two arguments ([§20.7.18](#)).

20.7.18 `public static int parseInt(String s, int radix)`
 throws `NumberFormatException`

The first argument is interpreted as representing a signed integer in the radix specified by the second argument. The components of the string must all be digits of the specified radix (as determined by whether `Character.digit` ([§20.5.23](#)) returns a nonnegative value), except that the first character may be '-' ('[\u002d](#)') to indicate a negative value. The resulting integer value is returned.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The `radix` is either smaller than `Character.MIN_RADIX` ([§20.5.3](#)) or larger than `Character.MAX_RADIX` ([§20.5.4](#)).
- Any character of the string is not a digit of the specified `radix`, except that the first character may be a minus sign `'-'` (`'\u002d'`) provided that the string is longer than length 1.
- The integer value represented by the string is not a value of type `int`.

Examples:

```
parseInt("0", 10) returns 0
parseInt("473", 10) returns 473
parseInt("-0", 10) returns 0
parseInt("-FF", 16) returns -255
parseInt("1100110", 2) returns 102
parseInt("2147483647", 10) returns 2147483647
parseInt("-2147483648", 10) returns -2147483648
parseInt("2147483648", 10) throws a NumberFormatException
parseInt("99", 8) throws a NumberFormatException
parseInt("Kona", 10) throws a NumberFormatException
parseInt("Kona", 27) returns 411787
```

20.7.19 `public static Integer valueOf(String s)`
 throws `NumberFormatException`

The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseInt` method that takes one argument ([§20.7.17](#)). The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s))
```

20.7.20 `public static Integer valueOf(String s, int radix)`
 throws `NumberFormatException`

The first argument is interpreted as representing a signed integer in the radix specified by the second argument, exactly as if the arguments were given to the `parseInt` method that takes two arguments ([§20.7.18](#)). The result is an `Integer` object that represents the integer value specified by the string.

In other words, this method returns an `Integer` object equal to the value of:

```
new Integer(Integer.parseInt(s, radix))
```

20.7.21 `public static Integer getInteger(String nm)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and an `Integer` object representing this value is returned. If there is no property of the specified name, or if the property does not have the correct numeric format, then `null` is returned.

In other words, this method returns an `Integer` object equal to the value of:

```
getInteger(nm, null)
```

20.7.22 `public static Integer getInteger(String nm, int val)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and an `Integer` object representing this value is returned. If the property does not have the correct numeric format, then an `Integer` object that represents the value of the second argument is returned.

In other words, this method returns an `Integer` object equal to the value of:

```
getInteger(nm, new Integer(val))
```

but in practice it may be implemented in a manner such as:

```
Integer result = getInteger(nm, null);  
return (result == null) ? new Integer(val) : result;
```

to avoid the unnecessary allocation of an `Integer` object when the default value is not needed.

20.7.23 `public static Integer getInteger(String nm, Integer val)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and an `Integer` object representing this value is returned.

- If the property value begins with the two ASCII characters `0x` or the ASCII character `#`, not followed by a minus sign, then the rest of it is parsed as a hexadecimal integer exactly as for the method `Integer.valueOf` (§20.7.20) with radix 16.
- If the property value begins with the ASCII character `0` followed by another character, it is parsed as an octal integer exactly as for the method `Integer.valueOf` (§20.7.20) with radix 8.
- Otherwise, the property value is parsed as a decimal integer exactly as for the method `Integer.valueOf` (§20.7.20) with radix 10.

The second argument serves as a default value. If there is no property of the specified name, or if the property does not have the correct numeric format, then the second argument is returned.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.8 The Class java.lang.Long

```
public final class Long extends Number {
    public static final long MIN_VALUE = 0x8000000000000000L;
    public static final long MAX_VALUE = 0x7fffffffffffffffffL;
    public Long(long value);
    public Long(String s)

        throws NumberFormatException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public int intValue();
    public long longValue();
    public float floatValue();
    public double doubleValue();
    public static String toString(long i);
    public static String toString(long i, int radix);
    public static String toHexString(long i);
    public static String toOctalString(long i);
    public static String toBinaryString(long i);
    public static long parseLong(String s)

        throws NumberFormatException;
    public static long parseLong(String s, int radix)

        throws NumberFormatException;
    public static Long valueOf(String s)

        throws NumberFormatException;
    public static Long valueOf(String s, int radix)

        throws NumberFormatException;
    public static Long getLong(String nm);
    public static Long getLong(String nm, long val);
    public static Long getLong(String nm, Long val);
}
```

20.8.1 `public static final long MIN_VALUE = 0x8000000000000000L;`

The constant value of this field is the lowest value of type `long`.

20.8.2 `public static final long MAX_VALUE = 0x7fffffffffffffffffL;`

The constant value of this field is the highest value of type `long`.

20.8.3 `public Long(long value)`

This constructor initializes a newly created `Long` object so that it represents the primitive value that is the argument.

20.8.4 `public Long(String s) throws NumberFormatException`

This constructor initializes a newly created `Long` object so that it represents the integer represented by the string in decimal form. The string is converted to a `long` value in exactly the manner used by the `parseLong` method ([§20.8.17](#)) for radix 10.

20.8.5 `public String toString()`

The `long` integer value represented by this `Long` object is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString` method that takes one argument ([§20.7.12](#)).

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.8.6 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Long` object that represents the same `long` value as this `Long` object.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

20.8.7 `public int hashCode()`

The result is the exclusive OR of the two halves of the primitive `long` value represented by this `Long` object. That is, the hashcode is the value of the expression:

```
(int) (this.longValue() ^ (this.longValue() >>> 32))
```

Overrides the `hashCode` method of `Object` ([§20.1.4](#)).

20.8.8 `public int intValue()`

The `long` value represented by this `Long` object is converted ([§5.1.3](#)) to type `int` and the result of the conversion is returned.

Overrides the `intValue` method of `Number` ([§20.6.1](#)).

20.8.9 `public long longValue()`

The `long` value represented by this `Long` object is returned.

Overrides the `longValue` method of `Number` ([§20.6.2](#)).

20.8.10 `public float floatValue()`

The `long` value represented by this `Long` object is converted ([§5.1.2](#)) to type `float` and the result of the conversion is returned.

Overrides the `floatValue` method of `Number` ([§20.6.3](#)).

20.8.11 `public double doubleValue()`

The `long` value represented by this `Long` object is converted ([§5.1.2](#)) to type `double` and the result of the conversion is returned.

Overrides the `doubleValue` method of `Number` ([§20.6.4](#)).

20.8.12 `public static String toString(long i)`

The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and the radix 10 were given as arguments to the `toString` method that takes two arguments ([§20.8.13](#)).

20.8.13 `public static String toString(long i, int radix)`

The first argument is converted to a signed representation in the radix specified by the second argument; this representation is returned as a string.

If the `radix` is smaller than `Character.MIN_RADIX` ([§20.5.3](#)) or larger than `Character.MAX_RADIX` ([§20.5.4](#)), then the value 10 is used instead.

If the first argument is negative, the first character of the result will be the character `'-'` (`'\u002d'`). If the first argument is not negative, no sign character appears in the result.

The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the magnitude will not be the zero character. The following ASCII characters are used as digits:

0123456789abcdefghijklmnopqrstuvwxyz

These are `'\u0030'` through `'\u0039'` and `'\u0061'` through `'\u007a'`. If the `radix` is *N*, then the first *N* of these characters are used as radix-*N* digits in the order shown. Thus, the digits for hexadecimal (radix 16) are 0123456789abcdef. If uppercase letters are desired, the `toUpperCase` method ([§20.12.36](#)) of class `String` may be called on the result:

`Long.toString(n, 16).toUpperCase()`

20.8.14 `public static String toHexString(long i)`

The argument is converted to an unsigned representation in hexadecimal radix (base 16); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {`ewc msdn cd, EWGraphic, LNG8x 0 /a "langref.BMP"`} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as hexadecimal digits:

0123456789abcdef

These are the characters '\u0030' through '\u0039' and '\u0061' through '\u0066'. If uppercase letters are desired, the `toUpperCase` method ([§20.12.36](#)) of class `String` may be called on the result:

```
Long.toHexString(n).toUpperCase()
```

20.8.15 `public static String toOctalString(long i)`

The argument is converted to an unsigned representation in octal radix (base 8); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {`ewc msdn`cd, `EWGraphic`, `LNG8x 1 /a "langref.BMP"`} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character '0' ('\u0030'); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The following characters are used as octal digits:

01234567

These are the characters '\u0030' through '\u0037'.

20.8.16 `public static String toBinaryString(long i)`

The argument is converted to an unsigned representation in binary radix (base 2); this representation is returned as a string.

The result represents the unsigned magnitude of the argument. This equals the argument plus {`ewc msdn`cd, `EWGraphic`, `LNG8x 2 /a "langref.BMP"`} if the argument is negative; otherwise, it equals the argument.

If the unsigned magnitude is zero, it is represented by a single zero character '0' ('\u0030'); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters '0' ('\u0030') and '1' ('\u0031') are used as binary digits.

20.8.17 `public static long parseLong(String s)`
throws `NumberFormatException`

The argument is interpreted as representing a signed decimal integer. The components of the string must all be decimal digits, except that the first character may be '-' ('\u002d') to indicate a negative value. The resulting `long` value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseLong` method that takes two arguments ([§20.8.18](#)).

Note that neither `L` nor `l` is permitted to appear at the end of the string as a type indicator, as would be permitted in Java source code ([§3.10.1](#)).

20.8.18 `public static long parseLong(String s, int radix)`

throws `NumberFormatException`

The first argument is interpreted as representing a signed integer in the radix specified by the second argument. The components of the string must all be digits of the specified radix (as determined by whether `Character.digit` (§20.5.23) returns a nonnegative value), except that the first character may be `'-'` (`'\u002d'`) to indicate a negative value. The resulting `long` value is returned.

Note that neither `L` nor `l` is permitted to appear at the end of the string as a type indicator, as would be permitted in Java source code (§3.10.1)—except that either `L` or `l` may appear as a digit for a radix greater than 22.

An exception of type `NumberFormatException` is thrown if any of the following situations occurs:

- The first argument is `null` or is a string of length zero.
- The radix is either smaller than `Character.MIN_RADIX` (§20.5.3) or larger than `Character.MAX_RADIX` (§20.5.4).
- The first character of the string is not a digit of the specified radix and is not a minus sign `'-'` (`'\u002d'`).
- The first character of the string is a minus sign and the string is of length 1.
- Any character of the string after the first is not a digit of the specified radix.
- The integer value represented by the string cannot be represented as a value of type `long`.

Examples:

```
parseLong("0", 10) returns 0L
parseLong("473", 10) returns 473L
parseLong("-0", 10) returns 0L
parseLong("-FF", 16) returns -255L
parseLong("1100110", 2) returns 102L
parseLong("99", 8) throws a NumberFormatException
parseLong("Hazelnut", 10) throws a NumberFormatException
parseLong("Hazelnut", 36) returns 1356099454469L
```

20.8.19 `public static Long valueOf(String s)`
throws `NumberFormatException`

The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the `parseLong` method that takes one argument (§20.8.17). The result is a `Long` object that represents the integer value specified by the string.

In other words, this method returns a `Long` object equal to the value of:

```
new Long(Long.parseLong(s))
```

20.8.20 `public static Long valueOf(String s, int radix)`
throws `NumberFormatException`

The first argument is interpreted as representing a signed integer in the radix specified by the second

argument, exactly as if the arguments were given to the `parseLong` method that takes two arguments ([§20.8.18](#)). The result is a `Long` object that represents the integer value specified by the string.

In other words, this method returns a `Long` object equal to the value of:

```
new Long(Long.parseLong(s, radix))
```

20.8.21 `public static Long getLong(String nm)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and a `Long` object representing this value is returned. If there is no property of the specified name, or if the property does not have the correct numeric format, then `null` is returned.

In other words, this method returns a `Long` object equal to the value of:

```
getLong(nm, null)
```

20.8.22 `public static Long getLong(String nm, long val)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and a `Long` object representing this value is returned. If there is no property of the specified name, or if the property does not have the correct numeric format, then a `Long` object that represents the value of the second argument is returned.

In other words, this method returns a `Long` object equal to the value of:

```
getLong(nm, new Long(val))
```

but in practice it may be implemented in a manner such as:

```
Long result = getLong(nm, null);  
return (result == null) ? new Long(val) : result;
```

to avoid the unnecessary allocation of a `Long` object when the default value is not needed.

20.8.23 `public static Long getLong(String nm, Long val)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (§20.18.9). The string value of this property is then interpreted as an integer value and a `Long` object representing this value is returned.

- If the property value begins with the two ASCII characters `0x` or the ASCII character `#`, not followed by a minus sign, then the rest of it is parsed as a hexadecimal integer exactly as for the

method `Long.valueOf` (§20.7.20) with radix 16.

- If the property value begins with the character 0 followed by another character, it is parsed as an octal integer exactly as for the method `Long.valueOf` (§20.7.20) with radix 8.
- Otherwise the property value is parsed as a decimal integer exactly as for the method `Long.valueOf` (§20.7.20) with radix 10.

Note that, in every case, neither `L` nor `l` is permitted to appear at the end of the property value as a type indicator, as would be permitted in Java source code (§3.10.1).

The second argument serves as a default value. If there is no property of the specified name, or if the property does not have the correct numeric format, then the second argument is returned.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.9 The Class java.lang.Float

```
public final class Float extends Number {
    public static final float MIN_VALUE = 1.4e-45f;
    public static final float MAX_VALUE = 3.4028235e+38f;
    public static final float NEGATIVE_INFINITY = -1.0f/0.0f;
    public static final float POSITIVE_INFINITY = 1.0f/0.0f;
    public static final float NaN = 0.0f/0.0f;
    public Float(float value);
    public Float(double value);
    public Float(String s)
        throws NumberFormatException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public int intValue();
    public long longValue();
    public float floatValue();
    public double doubleValue();
    public static String toString(float f);
    public static Float valueOf(String s)
        throws NullPointerException, NumberFormatException;
    public boolean isNaN();
    public static boolean isNaN(float v);
    public boolean isInfinite();
    public static boolean isInfinite(float v);
    public static int floatToIntBits(float value);
    public static float intBitsToFloat(int bits);
}
```

20.9.1 `public static final float MIN_VALUE = 1.4e-45f;`

The constant value of this field is the smallest positive nonzero value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x1)`.

20.9.2 `public static final float MAX_VALUE = 3.4028235e+38f;`

The constant value of this field is the largest positive finite value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7f7fffff)`.

20.9.3 `public static final float NEGATIVE_INFINITY =
-1.0f/0.0f;`

The constant value of this field is the negative infinity of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0xff800000)`.

20.9.4 `public static final float POSITIVE_INFINITY =
1.0f/0.0f;`

The constant value of this field is the positive infinity of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7f800000)`.

20.9.5 `public static final float NaN = 0.0f/0.0f;`

The constant value of this field is the Not-a-Number value of type `float`. It is equal to the value returned by `Float.intBitsToFloat(0x7fc00000)`.

20.9.6 `public Float(float value)`

This constructor initializes a newly created `Float` object so that it represents the primitive value that is the argument.

20.9.7 `public Float(double value)`

This constructor initializes a newly created `Float` object so that it represents the result of narrowing [\(§5.1.3\)](#) the argument from type `double` to type `float`.

20.9.8 `public Float(String s) throws NumberFormatException`

This constructor initializes a newly created `Float` object so that it represents the floating-point value of type `float` represented by the string. The string is converted to a `float` value in exactly the manner used by the `valueOf` method [\(§20.9.17\)](#).

20.9.9 `public String toString()`

The primitive `float` value represented by this `Float` object is converted to a string exactly as if by the method `toString` of one argument [\(§20.9.16\)](#).

Overrides the `toString` method of `Object` [\(§20.1.2\)](#).

20.9.10 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Float` object that represents the same `float` value as this `Float` object. For this purpose, two `float` values are considered to be the same if and only if the method `floatToIntBits` [\(§20.9.22\)](#) returns the same `int` value when applied to each. Note that even though the `==` operator returns `false` if both operands are `NaN`, this `equals` method will return `true` if this `Float` object and the argument are both `Float` objects that represent `NaN`. On the other hand, even though the `==` operator returns `true` if one operand is positive zero and the other is negative zero, this `equals` method will return `false` if this `Float` object and the argument represent zeroes of different sign. This definition allows hashtables to operate properly.

Overrides the `equals` method of `Object` [\(§20.1.3\)](#).

20.9.11 `public int hashCode()`

The result is the integer bit representation, exactly as produced by the method `floatToIntBits` [\(§20.9.22\)](#), of the primitive `float` value represented by this `Float` object.

Overrides the `hashCode` method of `Object` [\(§20.1.4\)](#).

20.9.12 `public int intValue()`

The `float` value represented by this `Float` object is converted ([§5.1.3](#)) to type `int` and the result of the conversion is returned.

Overrides the `intValue` method of `Number` ([§20.6.1](#)).

20.9.13 `public long longValue()`

The `float` value represented by this `Float` object is converted ([§5.1.3](#)) to type `long` and the result of the conversion is returned.

Overrides the `longValue` method of `Number` ([§20.6.2](#)).

20.9.14 `public float floatValue()`

The `float` value represented by this `Float` object is returned.

Overrides the `floatValue` method of `Number` ([§20.6.3](#)).

20.9.15 `public double doubleValue()`

The `float` value represented by this `Float` object is converted ([§5.1.2](#)) to type `double` and the result of the conversion is returned.

Overrides the `doubleValue` method of `Number` ([§20.6.4](#)).

20.9.16 `public static String toString(float f)`

The argument is converted to a readable string format as follows. All characters and characters in strings mentioned below are ASCII characters.

- If the argument is NaN, the result is the string "NaN".
- Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is '-' ('`\u002d`'); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
 - If *m* is infinity, it is represented by the characters "Infinity"; thus, positive infinity produces the result "Infinity" and negative infinity produces the result "-Infinity".
 - If *m* is zero, it is represented by the characters "0.0"; thus, negative zero produces the result "-0.0" and positive zero produces the result "0.0".
 - If *m* is greater than or equal to {ewc msdncl, EWGraphic, LNG9x 0 /a "langref.BMP"} but less than {ewc msdncl, EWGraphic, LNG9x 1 /a "langref.BMP"}, then it is represented as the integer part of *m*, in decimal form with no leading zeroes, followed by '.' ('`\u002E`'), followed by one or more decimal digits representing the fractional part of *m*.
 - If *m* is less than {ewc msdncl, EWGraphic, LNG9x 2 /a "langref.BMP"} or not less than {ewc msdncl, EWGraphic, LNG9x 3 /a "langref.BMP"}, then it is represented in so-called "computerized scientific notation." Let *n* be the unique integer such that {ewc msdncl, EWGraphic, LNG9x 4 /a "langref.BMP"}; then let *a* be the mathematically exact quotient of *m* and {ewc msdncl, EWGraphic, LNG9x 5 /a "langref.BMP"} so

that {ewc msdncl, EWGraphic, LNG9x 6 /a "langref.BMP"}. The magnitude is then represented as the integer part of *a*, as a single decimal digit, followed by ' . ' (\u002E), followed by decimal digits representing the fractional part of *a*, followed by the letter 'E' (\u0045), followed by a representation of *n* as a decimal integer, as produced by the method `Integer.toString` of one argument (§20.7.12).

How many digits must be printed for the fractional part of *m* or *a*? There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to uniquely distinguish the argument value from adjacent values of type `float`. That is, suppose that *x* is the exact mathematical value represented by the decimal representation produced by this method for a finite nonzero argument *f*. Then *f* must be the `float` value nearest to *x*; or, if two `float` values are equally close to *x*, then *f* must be one of them and the least significant bit of the significand of *f* must be 0.

[This specification for the method `toString` is scheduled for introduction in Java version 1.1. In previous versions of Java, this method produces `Inf` instead of `Infinity` for infinite values. Also, it renders finite values in the same form as the `%g` format of the `printf` function in the C programming language, which can lose precision because it produces at most six digits after the decimal point.]

20.9.17 `public static Float valueOf(String s)`
throws `NullPointerException`, `NumberFormatException`

The string *s* is interpreted as the representation of a floating-point value and a `Float` object representing that value is created and returned.

If *s* is `null`, then a `NullPointerException` is thrown.

Leading and trailing whitespace (§20.5.19) characters in *s* are ignored. The rest of *s* should constitute a *FloatValue* as described by the lexical syntax rules:

FloatValue:

*Sign*opt *Digits* . *Digit*s opt *ExponentPart*opt

*Sign*opt . *Digits* *ExponentPart*opt

where *Sign*, *Digits*, and *ExponentPart* are as defined in §3.10.2. If it does not have the form of a *FloatValue*, then a `NumberFormatException` is thrown. Otherwise, it is regarded as representing an exact decimal value in the usual "computerized scientific notation"; this exact decimal value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type `float` by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic. Finally, a new object of class `Float` is created to represent this `float` value.

Note that neither `F` nor `f` is permitted to appear in *s* as a type indicator, as would be permitted in Java source code (§3.10.1).

20.9.18 `public boolean isNaN()`

The result is `true` if and only if the value represented by this `Float` object is NaN.

20.9.19 `public static boolean isNaN(float v)`

The result is `true` if and only if the value of the argument is NaN.

20.9.20 `public boolean isInfinite()`

The result is `true` if and only if the value represented by this `Float` object is positive infinity or negative infinity.

20.9.21 `public static boolean isInfinite(float v)`

The result is `true` if and only if the value of the argument is positive infinity or negative infinity.

20.9.22 `public static int floatToIntBits(float value)`

The result is a representation of the floating-point argument according to the IEEE 754 floating-point "single format" bit layout:

- Bit 31 (the bit that is selected by the mask `0x80000000`) represents the sign of the floating-point number.
- Bits 30-23 (the bits that are selected by the mask `0x7f800000`) represent the exponent.
- Bits 22-0 (the bits that are selected by the mask `0x007fffff`) represent the significand (sometimes called the mantissa) of the floating-point number.
- If the argument is positive infinity, the result will be `0x7f800000`.
- If the argument is negative infinity, the result will be `0xff800000`.
- If the argument is NaN, the result will be `0x7fc00000`.

In all cases, the result is an integer that, when given to the `intBitsToFloat` method ([§20.9.23](#)), will produce a floating-point value equal to the argument to `floatToIntBits`.

20.9.23 `public static float intBitsToFloat(int bits)`

The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "single format" bit layout. That floating-point value is returned as the result.

- If the argument is `0x7f800000`, the result will be positive infinity.
- If the argument is `0xff800000`, the result will be negative infinity.
- If the argument is any value in the range `0x7f800001` through `0x7fffffff` or in the range `0xff800001` through `0xffffffff`, the result will be NaN. (All IEEE 754 NaN values are, in effect, lumped together by the Java language into a single value called NaN.)
- In all other cases, let *s*, *e*, and *m* be three values that can be computed from the argument:

```
int s = ((bits >> 31) == 0) ? 1 : -1;
int e = ((bits >> 23) & 0xff);
int m = (e == 0) ?
    (bits & 0x7fffff) << 1 :
    (bits & 0x7fffff) | 0x800000;
```


Then the floating-point result equals the value of the mathematical expression {ewc msdncd, EWGraphic, LNG9x 7 /a "langref.BMP"}.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.10 The Class java.lang.Double

```
public final class Double extends Number {
    public static final double MIN_VALUE =

        5e-324;
    public static final double MAX_VALUE =
        1.7976931348623157e+308;
    public static final double NEGATIVE_INFINITY = -1.0/0.0;
    public static final double POSITIVE_INFINITY = 1.0/0.0;
    public static final double NaN = 0.0/0.0;
    public Double(double value);
    public Double(String s)

        throws NumberFormatException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public int intValue();
    public long longValue();
    public float floatValue();
    public double doubleValue();
    public static String toString(double d);
    public static Double valueOf(String s)

        throws NullPointerException, NumberFormatException;
    public boolean isNaN();
    public static boolean isNaN(double v);
    public boolean isInfinite();
    public static boolean isInfinite(double v);
    public static long doubleToLongBits(double value);
    public static double longBitsToDouble(long bits);
}
```

20.10.1 `public static final double MIN_VALUE = 5e-324;`

The constant value of this field is the smallest positive nonzero value of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x1L)`.

20.10.2 `public static final double MAX_VALUE =`
`1.7976931348623157e+308;`

The constant value of this field is the largest positive finite value of type `double`. It is equal to the returned by:

`Double.longBitsToDouble(0x7fefffffffffffffL)`

20.10.3 `public static final double NEGATIVE_INFINITY = -1.0/0.0;`

The constant value of this field is the negative infinity of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0xfff0000000000000L)`.

20.10.4 `public static final double POSITIVE_INFINITY = 1.0/0.0;`

The constant value of this field is the positive infinity of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x7ff0000000000000L)`.

20.10.5 `public static final double NaN = 0.0/0.0;`

The constant value of this field is the Not-a-Number of type `double`. It is equal to the value returned by `Double.longBitsToDouble(0x7ff8000000000000L)`.

20.10.6 `public Double(double value)`

This constructor initializes a newly created `Double` object so that it represents the primitive value that is the argument.

20.10.7 `public Double(String s)`
throws `NumberFormatException`

This constructor initializes a newly created `Double` object so that it represents the floating-point value of type `double` represented by the string. The string is converted to a `double` value in exactly the manner used by the `valueOf` method ([§20.9.17](#)).

20.10.8 `public String toString()`

The primitive `double` value represented by this `Double` object is converted to a string exactly as if by the method `toString` of one argument ([§20.10.15](#)).

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.10.9 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Double` object that represents the same `double` value as this `Double` object. For this purpose, two `double` values are considered to be the same if and only if the method `doubleToLongBits` ([§20.10.21](#)) returns the same `long` value when applied to each. Note that even though the `==` operator returns `false` if both operands are `NaN`, this `equals` method will return `true` if this `Double` object and the argument are both `Double` objects that represent `NaN`. On the other hand, even though the `==` operator returns `true` if one operand is positive zero and the other is negative zero, this `equals` method will return `false` if this `Double` object and the argument represent zeroes of different sign. This allows hashtables to operate properly.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

20.10.10 `public int hashCode()`

The result is the exclusive OR of the two halves of the long integer bit representation, exactly as produced by the method `doubleToLongBits` ([§20.10.21](#)), of the primitive `double` value

represented by this `Double` object. That is, the hashCode is the value of the expression:

```
(int) (v ^ (v >>> 32))
```

where `v` is defined by:

```
long v = Double.doubleToLongBits(this.longValue());
```

Overrides the `hashCode` method of `Object` [\(§20.1.4\)](#).

20.10.11 `public int intValue()`

The `double` value represented by this `Double` object is converted [\(§5.1.3\)](#) to type `int` and the result of the conversion is returned.

Overrides the `intValue` method of `Number` [\(§20.6.1\)](#).

20.10.12 `public long longValue()`

The `double` value represented by this `Double` object is converted [\(§5.1.3\)](#) to type `long` and the result of the conversion is returned.

Overrides the `longValue` method of `Number` [\(§20.6.2\)](#).

20.10.13 `public float floatValue()`

The `double` value represented by this `Double` object is converted [\(§5.1.3\)](#) to type `float` and the result of the conversion is returned.

Overrides the `floatValue` method of `Number` [\(§20.6.3\)](#).

20.10.14 `public double doubleValue()`

The `double` value represented by this `Double` object is returned.

Overrides the `doubleValue` method of `Number` [\(§20.6.4\)](#).

20.10.15 `public static String toString(double d)`

The argument is converted to a readable string format as follows. All characters mentioned below are ASCII characters.

- If the argument is NaN, the result is the string "NaN".
- Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is '-' ('`\u002d`'); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
 - If *m* is infinity, it is represented by the characters "Infinity"; thus, positive infinity produces the result "Infinity" and negative infinity produces the result "-Infinity".
 - If *m* is zero, it is represented by the characters "0.0"; thus, negative zero produces the result "-0.0" and positive zero produces the result "0.0".

- If m is greater than or equal to `{ewc msdn cd, EWGraphic, LNG10x 0 /a "langref.BMP"}` but less than `{ewc msdn cd, EWGraphic, LNG10x 1 /a "langref.BMP"}`, then it is represented as the integer part of m , in decimal form with no leading zeroes, followed by `'.' (\u002E)`, followed by one or more decimal digits representing the fractional part of m .
- If m is less than `{ewc msdn cd, EWGraphic, LNG10x 2 /a "langref.BMP"}` or not less than `{ewc msdn cd, EWGraphic, LNG10x 3 /a "langref.BMP"}`, then it is represented in so-called "computerized scientific notation." Let n be the unique integer such that `{ewc msdn cd, EWGraphic, LNG10x 4 /a "langref.BMP"}`; then let a be the mathematically exact quotient of m and `{ewc msdn cd, EWGraphic, LNG10x 5 /a "langref.BMP"}` so that `{ewc msdn cd, EWGraphic, LNG10x 6 /a "langref.BMP"}`. The magnitude is then represented as the integer part of a , as a single decimal digit, followed by `'.' (\u002E)`, followed by decimal digits representing the fractional part of a , followed by the letter `'E' (\u0045)`, followed by a representation of n as a decimal integer, as produced by the method `Integer.toString` of one argument ([§20.7.12](#)).

How many digits must be printed for the fractional part of m or a ? There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to uniquely distinguish the argument value from adjacent values of type `double`. That is, suppose that x is the exact mathematical value represented by the decimal representation produced by this method for a finite nonzero argument d . Then d must be the `double` value nearest to x ; or if two `double` values are equally close to x , then d must be one of them and the least significant bit of the significand of d must be 0.

[This specification for the method `toString` is scheduled for introduction in Java version 1.1. In previous versions of Java, this method produces `Inf` instead of `Infinity` for infinite values. Also, it rendered finite values in the same form as the `%g` format of the `printf` function in the C programming language, which can lose information because it produces at most six digits after the decimal point.]

20.10.16 `public static Double valueOf(String s)`
throws `NullPointerException`, `NumberFormatException`

The string s is interpreted as the representation of a floating-point value and a `Double` object representing that value is created and returned.

If s is `null`, then a `NullPointerException` is thrown.

Leading and trailing whitespace ([§20.5.19](#)) characters in s are ignored. The rest of s should constitute a *FloatValue* as described by the lexical syntax rule:

FloatValue:

Signopt Digits . Digitsopt ExponentPartopt

Signopt . Digits ExponentPartopt

where *Sign*, *Digits*, and *ExponentPart* are as defined in [§3.10.2](#). If it does not have the form of a

FloatValue, then a `NumberFormatException` is thrown. Otherwise, it is regarded as representing an exact decimal value in the usual "computerized scientific notation"; this exact decimal value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type `double` by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic. Finally, a new object of class `Double` is created to represent the `double` value.

Note that neither `D` nor `d` is permitted to appear in `s` as a type indicator, as would be permitted in Java source code ([§3.10.1](#)).

20.10.17 `public boolean isNaN()`

The result is `true` if and only if the value represented by this `Double` object is NaN.

20.10.18 `public static boolean isNaN(double v)`

The result is `true` if and only if the value of the argument is NaN.

20.10.19 `public boolean isInfinite()`

The result is `true` if and only if the value represented by this `Double` object is positive infinity or negative infinity.

20.10.20 `public static boolean isInfinite(double v)`

The result is `true` if and only if the value of the argument is positive infinity or negative infinity.

20.10.21 `public static long doubleToLongBits(double value)`

The result is a representation of the floating-point argument according to the IEEE 754 floating-point "double format" bit layout:

- Bit 63 (the bit that is selected by the mask `0x8000000000000000L`) represents the sign of the floating-point number.
- Bits 62-52 (the bits that are selected by the mask `0x7ff0000000000000L`) represent the exponent.
- Bits 51-0 (the bits that are selected by the mask `0x000fffffffffffffffL`) represent the significand (sometimes called the mantissa) of the floating-point number.
- If the argument is positive infinity, the result will be `0x7ff0000000000000L`.
- If the argument is negative infinity, the result will be `0xfff0000000000000L`.
- If the argument is NaN, the result will be `0x7ff8000000000000L`.

In all cases, the result is a `long` integer that, when given to the `longBitsToDouble` method ([§20.10.22](#)), will produce a floating-point value equal to the argument to `doubleToLongBits`.

20.10.22 `public static double longBitsToDouble(long bits)`

The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "double format" bit layout. That floating-point value is returned as the result.

- If the argument is `0x7f80000000000000L`, the result will be positive infinity.

- If the argument is `0xff80000000000000L`, the result will be negative infinity.
- If the argument is any value in the range `0x7ff0000000000001L` through `0x7fffffffffffffffffL` or in the range `0xfff0000000000001L` through `0xffffffffffffffffL`, the result will be NaN. (All IEEE 754 NaN values are, in effect, lumped together by the Java language into a single value called NaN.)
- In all other cases, let s , e , and m be three values that can be computed from the argument:

```
int s = ((bits >> 63) == 0) ? 1 : -1;
int e = (int)((bits >> 52) & 0x7ffL);
long m = (e == 0) ?
    (bits & 0xffffffffffffffffL) << 1 :
    (bits & 0xffffffffffffffffL) | 0x10000000000000L;
```

Then the floating-point result equals the value of the mathematical expression `{ewc msdncd, EWGraphic, LNG10x 7 /a "langref.BMP"}`.

*Let beeves and home-bred kine partake
The sweets of Burn-mill meadow;
The swan on still St. Mary's Lake
Float double, swan and shadow!*
--William Wordsworth, *Yarrow Unvisited* (1803)

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

20.11 The Class java.lang.Math

*Oh, back to the days that were free from care in the 'Ology 'varsity shop,
With nothing to do but analyse air in an anemometrical top,
Or the differentiation of the trigonometrical pow'rs
Of the constant pi that made me sigh in those happy days of ours!*
--I. W. Litchfield, *Take Me Back to Tech* (1885)

The class `Math` contains useful basic numerical constants and methods.

```
public final class Math {
    public static final double E = 2.7182818284590452354;
    public static final double PI = 3.14159265358979323846;
    public static double sin(double a);
    public static double cos(double a);
    public static double tan(double a);
    public static double asin(double a);
    public static double acos(double a);
    public static double atan(double a);
    public static double atan2(double a, double b);
    public static double exp(double a);
    public static double log(double a);
    public static double sqrt(double a);
    public static double pow(double a, double b);
    public static double IEEEremainder(double f1, double f2);
    public static double ceil(double a);
    public static double floor(double a);
    public static double rint(double a);
    public static int round(float a);
    public static long round(double a);
    public static double random();
    public static int abs(int a);
    public static long abs(long a);
    public static float abs(float a);
    public static double abs(double a);
    public static int min(int a, int b);
    public static long min(long a, long b);
    public static float min(float a, float b);
    public static double min(double a, double b);
    public static int max(int a, int b);
    public static long max(long a, long b);
    public static float max(float a, float b);
    public static double max(double a, double b);
}
```

To ensure portability of Java programs, the specifications of many of the numerical functions in this package require that they produce the same results as certain published algorithms. These algorithms are available from the well-known network library `netlib` as the package `fdlibm` ("Freely Distributable Math Library"). These algorithms, which are written in the C programming language, are to be understood as if executed in Java execution order with all floating-point operations following the rules of Java floating-point arithmetic.

The network library may be found at <http://netlib.att.com> on the World Wide Web; then

perform a keyword search for `fdlibm`. The library may also be retrieved by E-mail; to begin the process, send a message containing the line:

```
send index from fdlibm
```

to `netlib@research.att.com`. The Java math library is defined with respect to the version of `fdlibm` dated 95/01/04. Where `fdlibm` provides more than one definition for a function (such as `acos`), the "IEEE754 core function" version is to be used (residing in a file whose name begins with the letter `e`).

A complete and self-contained description of the algorithms to be used for these functions will be provided in a future version of this specification. It is also anticipated that the algorithms will be coded in Java to provide a reference implementation that is not tied to `fdlibm`.

20.11.1 `public static final double E = 2.7182818284590452354;`

The constant value of this field is the `double` value that is closer than any other to `e`, the base of the natural logarithms.

20.11.2 `public static final double PI = 3.14159265358979323846;`

The constant value of this field is the `double` value that is closer than any other to `{ewc msdn cd, EWGraphic, LNG11x 0 /a "langref.BMP"}`, the ratio of the circumference of a circle to its diameter.

20.11.3 `public static double sin(double a)`

This method computes an approximation to the sine of the argument, using the `sin` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero.

20.11.4 `public static double cos(double a)`

This method computes an approximation to the cosine of the argument, using the `cos` algorithm as published in `fdlibm` (see the introduction to this section).

Special case:

- If the argument is NaN or an infinity, then the result is NaN.

20.11.5 `public static double tan(double a)`

This method computes an approximation to the tangent of the argument, using the `tan` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero.

20.11.6 `public static double asin(double a)`

This method computes an approximation to the arc sine of the argument, using the `asin` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero.

20.11.7 `public static double acos(double a)`

This method computes an approximation to the arc cosine of the argument, using the `acos` algorithm as published in `fdlibm` (see the introduction to this section).

Special case:

- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.

20.11.8 `public static double atan(double a)`

This method computes an approximation to the arc tangent of the argument, using the `atan` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive zero, then the result is positive zero; if the argument is negative zero, then the result is negative zero.

20.11.9 `public static double atan2(double y, double x)`

This method computes an approximation to the arc tangent of the quotient {ewc msdn cd, EWGraphic, LNG11x 1 /a "langref.BMP"} of the arguments, using the `atan2` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If either argument is NaN, then the result is NaN.
- If the first argument is positive zero and the second argument is positive, or the first argument is positive and finite and the second argument is positive infinity, then the result is positive zero.
- If the first argument is negative zero and the second argument is positive, or the first argument is

negative and finite and the second argument is positive infinity, then the result is negative zero.

- If the first argument is positive zero and the second argument is negative, or the first argument is positive and finite and the second argument is negative infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 2 /a "langref.BMP"}.
- If the first argument is negative zero and the second argument is negative, or the first argument is negative and finite and the second argument is negative infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 3 /a "langref.BMP"}.
- If the first argument is positive and the second argument is positive zero or negative zero, or the first argument is positive infinity and the second argument is finite, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 4 /a "langref.BMP"}.
- If the first argument is negative and the second argument is positive zero or negative zero, or the first argument is negative infinity and the second argument is finite, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 5 /a "langref.BMP"}.
- If both arguments are positive infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 6 /a "langref.BMP"}.
- If the first argument is positive infinity and the second argument is negative infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 7 /a "langref.BMP"}.
- If the first argument is negative infinity and the second argument is positive infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 8 /a "langref.BMP"}.
- If both arguments are negative infinity, then the result is the `double` value closest to {ewc msdn cd, EWGraphic, LNG11x 9 /a "langref.BMP"}.

20.11.10 `public static double exp(double a)`

This method computes an approximation to the exponential function of the argument (e raised to the power of the argument, where e is the base of the natural logarithms ([§20.11.1](#))), using the `exp` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is negative infinity, then the result is positive zero.

20.11.11 `public static double log(double a)`

This method computes an approximation to the natural logarithm of the argument, using the `log` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is negative infinity.

20.11.12 `public static double sqrt(double a)`

Whan that Aprill with his shoures soote

The droghte of March hath perced to the roote . . .

--Geoffrey Chaucer (1328-1400), *The Canterbury Tales*, General Prologue

This method computes an approximation to the square root of the argument.

Special cases:

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Otherwise, the result is the `double` value closest to the true mathematical square root of the argument value.

20.11.13 `public static double pow(double a, double b)`

This method computes an approximation to the mathematical operation of raising the first argument to the power of the second argument, using the `pow` algorithm as published in `fdlibm` (see the introduction to this section).

Special cases:

- If the second argument is positive or negative zero, then the result is `1.0`.
- If the second argument is `1.0`, then the result is the same as the first argument.
- If the second argument is NaN, then the result is NaN.
- If the first argument is NaN and the second argument is nonzero, then the result is NaN.
- If the absolute value of the first argument is greater than 1 and the second argument is positive infinity, or the absolute value of the first argument is less than 1 and the second argument is negative infinity, then the result is positive infinity.
- If the absolute value of the first argument is greater than 1 and the second argument is negative infinity, or the absolute value of the first argument is less than 1 and the second argument is positive infinity, then the result is positive zero.
- If the absolute value of the first argument equals 1 and the second argument is infinite, then the result is NaN.
- If the first argument is positive zero and the second argument is greater than zero, or the first argument is positive infinity and the second argument is less than zero, then the result is positive zero.
- If the first argument is positive zero and the second argument is less than zero, or the first argument is positive infinity and the second argument is greater than zero, then the result is positive infinity.
- If the first argument is negative zero and the second argument is greater than zero but not a finite odd integer, or the first argument is negative infinity and the second argument is less than zero but not a finite odd integer, then the result is positive zero.
- If the first argument is negative zero and the second argument is a positive finite odd integer, or the first argument is negative infinity and the second argument is a negative finite odd integer, then the result is negative zero.

- If the first argument is negative zero and the second argument is less than zero but not a finite odd integer, or the first argument is negative infinity and the second argument is greater than zero but not a finite odd integer, then the result is positive infinity.
- If the first argument is negative zero and the second argument is a negative finite odd integer, or the first argument is negative infinity and the second argument is a positive finite odd integer, then the result is negative infinity.
- If the first argument is less than zero and the second argument is a finite even integer, then the result is equal to the result of raising the absolute value of the first argument to the power of the second argument.
- If the first argument is less than zero and the second argument is a finite odd integer, then the result is equal to the negative of the result of raising the absolute value of the first argument to the power of the second argument.
- If the first argument is finite and less than zero and the second argument is finite and not an integer, then the result is NaN.
- If both arguments are integers, then the result is exactly equal to the mathematical result of raising the first argument to the power of the second argument if that result can in fact be represented exactly as a `double` value.

(In the foregoing descriptions, a floating-point value is considered to be an integer if and only if it is a fixed point of the method `ceil` (§20.11.15) or, which is the same thing, a fixed point of the method `floor` (§20.11.16). A value is a fixed point of a one-argument method if and only if the result of applying the method to the value is equal to the value.)

20.11.14 `public static double IEEEremainder(double x, double y)`

This method computes the remainder operation on two arguments as prescribed by the IEEE 754 standard: the remainder value is mathematically equal to $\{ewc\ msdncd, EWGraphic, LNG11x\ 10 /a\ "langref.BMP"\}$ where $\{ewc\ msdncd, EWGraphic, LNG11x\ 11 /a\ "langref.BMP"\}$ is the mathematical integer closest to the exact mathematical value of the quotient $\{ewc\ msdncd, EWGraphic, LNG11x\ 12 /a\ "langref.BMP"\}$; if two mathematical integers are equally close to $\{ewc\ msdncd, EWGraphic, LNG11x\ 13 /a\ "langref.BMP"\}$ then n is the integer that is even. If the remainder is zero, its sign is the same as the sign of the first argument.

Special cases:

- If either argument is NaN, or the first argument is infinite, or the second argument is positive zero or negative zero, then the result is NaN.
- If the first argument is finite and the second argument is infinite, then the result is the same as the first argument.

20.11.15 `public static double ceil(double a)`

The result is the smallest (closest to negative infinity) `double` value that is not less than the argument and is equal to a mathematical integer.

Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

- If the argument value is less than zero but greater than -1.0 , then the result is negative zero.

Note that the value of `Math.ceil(x)` is exactly the value of `-Math.floor(-x)`.

20.11.16 `public static double floor(double a)`

The result is the largest (closest to positive infinity) `double` value that is not greater than the argument and is equal to a mathematical integer.

Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

20.11.17 `public static double rint(double a)`

The result is the `double` value that is closest in value to the argument and is equal to a mathematical integer. If two `double` values that are mathematical integers are equally close to the value of the argument, the result is the integer value that is even.

Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

20.11.18 `public static int round(float a)`

Round numbers are always false.

--Samuel Johnson (1709-1784)

The result is rounded to an integer by adding {ewc msdncd, EWGraphic, LNG11x 14 /a "langref.BMP"}, taking the floor of the result, and casting the result to type `int`.

In other words, the result is equal to the value of the expression:

```
(int)Math.floor(a + 0.5f)
```

Special cases:

- If the argument is NaN, the result is 0.
- If the argument is negative infinity, or indeed any value less than or equal to the value of `Integer.MIN_VALUE` ([§20.7.1](#)), the result is equal to the value of `Integer.MIN_VALUE`.
- If the argument is positive infinity, or indeed any value greater than or equal to the value of `Integer.MAX_VALUE` ([§20.7.2](#)), the result is equal to the value of `Integer.MAX_VALUE`.

20.11.19 `public static long round(double a)`

The result is rounded to an integer by adding `{ewc msdncl, EWGraphic, LNG11x 15 /a "langref.BMP"}`, taking the floor of the result, and casting the result to type `long`.

In other words, the result is equal to the value of the expression:

```
(long)Math.floor(a + 0.5d)
```

Special cases:

- If the argument is NaN, the result is 0.
- If the argument is negative infinity, or indeed any value less than or equal to the value of `Long.MIN_VALUE` ([§20.7.1](#)), the result is equal to the value of `Long.MIN_VALUE`.
- If the argument is positive infinity, or indeed any value greater than or equal to the value of `Long.MAX_VALUE` ([§20.7.2](#)), the result is equal to the value of `Long.MAX_VALUE`.

20.11.20 `public static double random()`

The result is a double value with positive sign, greater than or equal to zero but less than 1.0, chosen pseudorandomly with (approximately) uniform distribution from that range.

When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression

```
new java.util.Random()
```

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else.

This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom number generator.

20.11.21 `public static int abs(int a)`

The result is the absolute value of the argument, if possible.

If the argument is not negative, the argument is returned.

If the argument is negative, the negation of the argument is returned. Note that if the argument is equal to the value of `Integer.MIN_VALUE` ([§20.7.1](#)), the most negative representable `int` value, the result will be that same negative value.

20.11.22 `public static long abs(long a)`

The result is the absolute value of the argument, if possible.

If the argument is not negative, the argument is returned.

If the argument is negative, the negation of the argument is returned. Note that if the argument is equal to the value of `Long.MIN_VALUE` ([§20.8.1](#)), the most negative representable `long` value, the result will be that same negative value.

20.11.23 `public static float abs(float a)`

The argument is returned with its sign changed to be positive.

Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is equal to the value of the expression:

```
Float.intBitsToFloat(0x7fffffff & Float.floatToIntBits(a))
```

[This specification for the method `abs` is scheduled for introduction in Java version 1.1. In previous versions of Java, `abs(-0.0f)` returns `-0.0f`, which is not correct.]

20.11.24 `public static double abs(double a)`

The argument is returned with its sign changed to be positive.

Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is equal to the value of the expression:

```
Double.longBitsToDouble((Double.doubleToLongBits(a) << 1) >>> 1)
```

[This specification for the method `abs` is scheduled for introduction in Java version 1.1. In previous versions of Java, `abs(-0.0d)` returns `-0.0d`, which is not correct.]

20.11.25 `public static int min(int a, int b)`

E duobus malis minimum eligendum.

--Marcus Tullius Cicero (106-43 B. C.), *De officiis*, iii

The result is the smaller of the two arguments—that is, the one closer to the value of `Integer.MIN_VALUE` ([§20.7.1](#)). If the arguments have the same value, the result is that same value.

20.11.26 `public static long min(long a, long b)`

Of harmes two the lesse is for to cheese.

--Geoffrey Chaucer (1328-1400), *Troilus and Criseyde*, Book ii

The result is the smaller of the two arguments-that is, the one closer to the value of `Long.MIN_VALUE` ([§20.8.1](#)). If the arguments have the same value, the result is that same value.

20.11.27 `public static float min(float a, float b)`

Of two evils, the less is always to be chosen.

--Thomas a Kempis (1380-1471), *Imitation of Christ*, Book iii, chapter 12

The result is the smaller of the two arguments-that is, the one closer to negative infinity. If the arguments have the same value, the result is that same value.

Special cases:

- If one argument is positive zero and the other is negative zero, the result is negative zero.
- If either argument is NaN, the result is NaN.

[This specification for the method `min` is scheduled for introduction in Java version 1.1. In previous versions of Java, `min(0.0f, -0.0f)` returns `0.0f`, which is not correct.]

20.11.28 `public static double min(double a, double b)`

Of two evils I have chose the least.

--Matthew Prior (1664-1721), *Imitation of Horace*

The result is the smaller of the two arguments-that is, the one closer to negative infinity. If the arguments have the same value, the result is that same value.

Special cases:

- If one argument is positive zero and the other is negative zero, the result is negative zero.
- If either argument is NaN, the result is NaN.

[This specification for the method `min` is scheduled for introduction in Java version 1.1. In previous versions of Java, `min(0.0d, -0.0d)` returns `0.0d`, which is not correct.]

20.11.29 `public static int max(int a, int b)`

The result is the larger of the two arguments-that is, the one closer to the value of `Integer.MAX_VALUE` ([§20.7.2](#)). If the arguments have the same value, the result is that same value.

20.11.30 `public static long max(long a, long b)`

The result is the larger of the two arguments-that is, the one closer to the value of `Long.MAX_VALUE` (§20.8.2). If the arguments have the same value, the result is that same value.

20.11.31 `public static float max(float a, float b)`

The result is the larger of the two arguments-that is, the one closer to positive infinity. If the arguments have the same value, the result is that same value.

Special cases:

- If one argument is positive zero and the other is negative zero, the result is positive zero.
- If either argument is NaN, the result is NaN.

[This specification for the method `max` is scheduled for introduction in Java version 1.1. In previous versions of Java, `max(-0.0f, 0.0f)` returns `-0.0f`, which is not correct.]

20.11.32 `public static double max(double a, double b)`

The result is the larger of the two arguments-that is, the one closer to positive infinity. If the arguments have the same value, the result is that same value.

Special cases:

- If one argument is positive zero and the other is negative zero, the result is positive zero.
- If either argument is NaN, the result is NaN.

[This specification for the method `max` is scheduled for introduction in Java version 1.1. In previous versions of Java, `max(-0.0d, 0.0d)` returns `-0.0d`, which is not correct.].

*In mathematics he was greater
Than Tycho Brahe or Erra Pater
For he, by geometric scale,
Could take the size of pots of ale;
Resolve, by sines and tangents straight
Whether bread or butter wanted weight;
And wisely tell what hour o' the day
The clock does strike, by algebra.
--Samuel Butler, *Hudibras*, Part I, canto i*

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.12 The Class java.lang.String

An object of type `String`, once created, is immutable. It represents a fixed-length sequence of characters. Compare this to the class `StringBuffer` ([§20.13](#)), which represents a modifiable, variable-length sequence of characters.

The class `String` has methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, for creating a copy of a string with all characters translated to uppercase or to lowercase, and so on.

```
public final class String {
    public String();
    public String(String value)
        throws NullPointerException;
    public String(StringBuffer buffer)
        throws NullPointerException;
    public String(char[] value)
        throws NullPointerException;
    public String(char[] value, int offset, int count)
        throws NullPointerException, IndexOutOfBoundsException;
    public String(byte[] ascii, int hibyte)
        throws NullPointerException;
    public String(byte[] ascii, int hibyte,
        int offset, int count)
        throws NullPointerException, IndexOutOfBoundsException;
    public String toString();
    public boolean equals(Object anObject);
    public int hashCode();
    public int length();
    public char charAt(int index);
    public void getChars(int srcBegin, int srcEnd,
        char dst[], int dstBegin)

        throws NullPointerException, IndexOutOfBoundsException;
    public void getBytes(int srcBegin, int srcEnd,
        byte dst[], int dstBegin)
        throws NullPointerException, IndexOutOfBoundsException;
    public char[] toCharArray();
    public boolean equalsIgnoreCase(String anotherString);
    public int compareTo(String anotherString)
        throws NullPointerException;
    public boolean regionMatches(int toffset, String other,

        int ooffset, int len)
        throws NullPointerException;
    public boolean regionMatches(boolean ignoreCase, int toffset,

        String other, int ooffset, int len)
        throws NullPointerException;
    public boolean startsWith(String prefix)
        throws NullPointerException;
    public boolean startsWith(String prefix, int toffset)
        throws NullPointerException;
    public boolean endsWith(String suffix)
```



```

        throws NullPointerException;
    public int indexOf(int ch);
    public int indexOf(int ch, int fromIndex);
    public int indexOf(String str)
        throws NullPointerException;
    public int indexOf(String str, int fromIndex)
        throws NullPointerException;
    public int lastIndexOf(int ch);
    public int lastIndexOf(int ch, int fromIndex);
    public int lastIndexOf(String str)
        throws NullPointerException;
    public int lastIndexOf(String str, int fromIndex)
        throws NullPointerException;
    public String substring(int beginIndex);
    public String substring(int beginIndex, int endIndex);
    public String concat(String str)
        throws NullPointerException;
    public String replace(char oldChar, char newChar);
    public String toLowerCase();
    public String toUpperCase();
    public String trim();
    public static String valueOf(Object obj);
    public static String valueOf(char[] data)
        throws NullPointerException;
    public static String valueOf(char[] data,
        int offset, int count)
        throws NullPointerException, IndexOutOfBoundsException;
    public static String valueOf(boolean b);
    public static String valueOf(char c);
    public static String valueOf(int i);
    public static String valueOf(long l);
    public static String valueOf(float f);
    public static String valueOf(double d);
    public String intern();
}

```

20.12.1 `public String()`

This constructor initializes a newly created `String` object so that it represents an empty character sequence.

20.12.2 `public String(String value)`

This constructor initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

20.12.3 `public String(StringBuffer buffer)` `throws NullPointerException`

This constructor initializes a newly created `String` object so that it represents the sequence of characters that is currently contained in the `StringBuffer` argument ([§20.13](#)). The contents of the string buffer are copied; subsequent modification of the string buffer does not affect the newly

created string.

If `buffer` is null, then a `NullPointerException` is thrown.

20.12.4 `public String(char[] data)`
throws `NullPointerException`

This constructor initializes a newly created `String` object so that it represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

If `data` is null, then a `NullPointerException` is thrown.

20.12.5 `public String(char[] data, int offset, int count)`
throws `NullPointerException`, `IndexOutOfBoundsException`

This constructor initializes a newly created `String` object so that it represents the sequence of characters currently contained in a subarray of the character array argument. The `offset` argument is the index of the first character of the subarray and the `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

If `data` is null, then a `NullPointerException` is thrown.

If `offset` is negative, or `count` is negative, or `offset+count` is larger than `data.length`, then an `IndexOutOfBoundsException` is thrown.

20.12.6 `public String(byte[] ascii, int hibyte)`
throws `NullPointerException`

This constructor initializes a newly created `String` object so that it represents a sequence of characters constructed from an array of 8-bit integer values. Each character `c` in the result string is constructed from the corresponding element `b` of the byte array in such a way that:

$$c == ((hibyte \& 0xff) \ll 8) | (b \& 0xff)$$

If `ascii` is null, then a `NullPointerException` is thrown.

20.12.7 `public String(byte[] ascii, int hibyte,`
 `int offset, int count)`
throws `NullPointerException`,
 `IndexOutOfBoundsException`

This constructor initializes a newly created `String` object so that it represents the sequence of characters constructed from a subarray of an array of 8-bit integer values. The `offset` argument is the index of the first byte of the subarray and the `count` argument specifies the length of the subarray. Each character `c` in the result string is constructed from the corresponding element `b` of the byte subarray in such a way that:

$$c == ((hibyte \& 0xff) \ll 8) | (b \& 0xff)$$

If `ascii` is null, then a `NullPointerException` is thrown.

If `offset` is negative, or `count` is negative, or `offset+count` is larger than `ascii.length`, then an `IndexOutOfBoundsException` is thrown.

20.12.8 `public String toString()`

A reference to this object (which is, after all, already a `String`) is returned.

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.12.9 `public boolean equals(Object anObject)`

The result is `true` if and only if the argument is not null and is a `String` object that represents the same sequence of characters as this `String` object.

Overrides the `equals` method of `Object` ([§20.1.3](#)).

See also the methods `equalsIgnoreCase` ([§20.12.16](#)) and `compareTo` ([§20.12.17](#)).

20.12.10 `public int hashCode()`

The hashcode for a `String` object is computed in one of two ways, depending on its length. Let n be the length ([§20.12.11](#)) of the character sequence and let $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 0\ /\ a\ "langref.BMP"\}$ mean the character with index i .

- If $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 1\ /\ a\ "langref.BMP"\}$, then the hashcode is computed as $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 2\ /\ a\ "langref.BMP"\}$ using `int` arithmetic.
- If $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 3\ /\ a\ "langref.BMP"\}$, then the hashcode is computed as $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 4\ /\ a\ "langref.BMP"\}$ using `int` arithmetic, where $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 5\ /\ a\ "langref.BMP"\}$ and $\{ewc\ msdn\ cd, EWGraphic, LNG12x\ 6\ /\ a\ "langref.BMP"\}$, sampling only eight or nine characters of the string.

Overrides the `hashCode` method of `Object` ([§20.1.4](#)).

20.12.11 `public int length()`

The length of the sequence of characters represented by this `String` object is returned.

20.12.12 `public char charAt(int index)`
throws `IndexOutOfBoundsException`

This method returns the character indicated by the `index` argument within the sequence of characters represented by this `String`. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing. If the `index` argument is negative or not less than the length ([§20.12.11](#)) of this string, then an `IndexOutOfBoundsException` is thrown.

20.12.13 `public void getChars(int srcBegin, int srcEnd,`


```
        char dst[], int dstBegin)
throws NullPointerException,
    IndexOutOfBoundsException
```

Characters are copied from this `String` object into the destination character array `dst`. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus the total number of characters to be copied is `srcEnd-srcBegin`). The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index `dstBegin+(srcEnd-srcBegin)-1`.

If `dst` is null, then a `NullPointerException` is thrown.

An `IndexOutOfBoundsException` is thrown if any of the following is true:

- `srcBegin` is negative
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this `String`
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

```
20.12.14    public void getBytes(int srcBegin, int srcEnd,
        byte dst[], int dstBegin)
throws NullPointerException,
    IndexOutOfBoundsException
```

Characters are copied from this `String` object into the destination byte array `dst`. Each byte receives only the eight low-order bits of the corresponding character. The eight high-order bits of each character are not copied and do not participate in the transfer in any way. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus the total number of characters to be copied is `srcEnd-srcBegin`). The characters, converted to bytes, are copied into the subarray of `dst` starting at index `dstBegin` and ending at index `dstBegin+(srcEnd-srcBegin)-1`.

If `dst` is null, then a `NullPointerException` is thrown.

An `IndexOutOfBoundsException` is thrown if any of the following is true:

- `srcBegin` is negative
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this `String`
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

```
20.12.15    public char[] toCharArray()
```

A new character array is created and returned. The length of the array is equal to the length ([§20.12.11](#)) of this `String` object. The array is initialized to contain the character sequence represented by this `String` object.

```
20.12.16    public boolean equalsIgnoreCase(String anotherString)
```


The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this `String` object, where case is ignored.

Two characters are considered the same, ignoring case, if at least one of the following is true:

- The two characters are the same (as compared by the `==` operator).
- Applying the method `Character.toUpperCase` (§20.5.21) to each character produces the same result.
- Applying the method `Character.toLowerCase` (§20.5.20) to each character produces the same result.

Two sequences of characters are the same, ignoring case, if the sequences have the same length and corresponding characters are the same, ignoring case.

See also the method `equals` (§20.12.9).

20.12.17 `public int compareTo(String anotherString)`
throws `NullPointerException`

The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns 0 exactly when the `equals` method (§20.12.9) would return `true`.

If `anotherString` is `null`, then a `NullPointerException` is thrown.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the `<` operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position *k* in the two strings- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings-that is, the value:

```
this.length()-anotherString.length()
```

20.12.18 `public boolean regionMatches(int toffset,`
 `String other, int ooffset, int len)`
throws `NullPointerException`

A substring of this `String` object is compared to a substring of the argument `other`. The result is `true` if these substrings represent identical character sequences. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of `other` to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least

one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the `other` argument.
- There is some nonnegative integer k less than `len` such that:`this.charAt(toffset+k) != other.charAt(ooffset+k)`

If `other` is null, then a `NullPointerException` is thrown.

```
20.12.19    public boolean regionMatches(boolean ignoreCase,
        int toffset, String other, int ooffset, int len)
throws NullPointerException
```

A substring of this `String` object is compared to a substring of the argument `other`. The result is `true` if these substrings represent character sequences that are the same, ignoring case if and only if `ignoreCase` is `true`. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of `other` to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the `other` argument.
- There is some nonnegative integer `k` less than `len` such that:`this.charAt(toffset+k) != other.charAt(ooffset+k)`
- `ignoreCase` is `true` and there is some nonnegative integer `k` less than `len` such that:

```
Character.toLowerCase(this.charAt(toffset+k)) !=
    Character.toLowerCase(other.charAt(ooffset+k))
```

and:

```
Character.toUpperCase(this.charAt(toffset+k)) !=
    Character.toUpperCase(other.charAt(ooffset+k))
```

If `other` is null, then a `NullPointerException` is thrown.

```
20.12.20    public boolean startsWith(String prefix)
throws NullPointerException
```

The result is `true` if and only if the character sequence represented by the argument is a prefix of the character sequence represented by this `String` object.

If `prefix` is null, a `NullPointerException` is thrown.

Note that the result will be `true` if the argument is an empty string or is equal to this `String` object as determined by the `equals` method ([§20.12.9](#)).

20.12.21 `public boolean startsWith(String prefix, int toffset)`
throws `NullPointerException`

The result is `true` if and only if the character sequence represented by the argument is a prefix of the substring of this `String` object starting at index `toffset`.

If `prefix` is null, then a `NullPointerException` is thrown.

The result is `false` if `toffset` is negative or greater than the length of this `String` object; otherwise, the result is the same as the result of the expression

```
this.substring(toffset).startsWith(prefix)
```

20.12.22 `public boolean endsWith(String suffix)`
throws `NullPointerException`

The result is `true` if and only if the character sequence represented by the argument is a suffix of the character sequence represented by this `String` object.

If `suffix` is null, then a `NullPointerException` is thrown.

Note that the result will be `true` if the argument is an empty string or is equal to this `String` object as determined by the `equals` method ([§20.12.9](#)).

20.12.23 `public int indexOf(int ch)`

If a character with value `ch` occurs in the character sequence represented by this `String` object, then the index of the first such occurrence is returned—that is, the smallest value `k` such that:

```
this.charAt(k) == ch
```

is `true`. If no such character occurs in this string, then `-1` is returned.

20.12.24 `public int indexOf(int ch, int fromIndex)`

If a character with value `ch` occurs in the character sequence represented by this `String` object at an index no smaller than `fromIndex`, then the index of the first such occurrence is returned—that is, the smallest value `k` such that:

```
(this.charAt(k) == ch) && (k >= fromIndex)
```

is `true`. If no such character occurs in this string at or after position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were

zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: -1 is returned.

20.12.25 `public int indexOf(String str)`
throws `NullPointerException`

If the string `str` occurs as a substring of this `String` object, then the index of the first character of the first such substring is returned-that is, the smallest value k such that:

`this.startsWith(str, k)`

is true. If `str` does not occur as a substring of this string, then -1 is returned.

If `str` is null, a `NullPointerException` is thrown.

20.12.26 `public int indexOf(String str, int fromIndex)`
throws `NullPointerException`

If the string `str` occurs as a substring of this `String` object starting at an index no smaller than `fromIndex`, then the index of the first character of the first such substring is returned-that is, the smallest value k such that:

`this.startsWith(str, k) && (k >= fromIndex)`

is true. If `str` does not occur as a substring of this string at or after position `fromIndex`, then -1 is returned.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: -1 is returned.

If `str` is null, a `NullPointerException` is thrown.

20.12.27 `public int lastIndexOf(int ch)`

If a character with value `ch` occurs in the character sequence represented by this `String` object, then the index of the last such occurrence is returned-that is, the largest value k such that:

`this.charAt(k) == ch`

is true. If no such character occurs in this string, then -1 is returned.

20.12.28 `public int lastIndexOf(int ch, int fromIndex)`

If a character with value `ch` occurs in the character sequence represented by this `String` object at an index no larger than `fromIndex`, then the index of the last such occurrence is returned-that is, the largest value k such that:


```
(this.charAt(k) == ch) && (k <= fromIndex)
```

is true. If no such character occurs in this string at or before position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is greater than or equal to the length of this string, it has the same effect as if it were equal to one less than the length of this string: this entire string may be searched. If it is negative, it has the same effect as if it were `-1`: `-1` is returned.

20.12.29 `public int lastIndexOf(String str)`
throws `NullPointerException`

If the string `str` occurs as a substring of this `String` object, then the index of the first character of the last such substring is returned—that is, the largest value `k` such that:

```
this.startsWith(str, k)
```

is true. If `str` does not occur as a substring of this string, then `-1` is returned.

If `str` is null, a `NullPointerException` is thrown.

20.12.30 `public int lastIndexOf(String str, int fromIndex)`
throws `NullPointerException`

If the string `str` occurs as a substring of this `String` object starting at an index no larger than `fromIndex`, then the index of the first character of the last such substring is returned—that is, the largest value `k` such that:

```
this.startsWith(str, k) && (k <= fromIndex)
```

is true. If `str` does not occur as a substring of this string at or before position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: this entire string may be searched. If it is negative, it has the same effect as if it were `-1`: `-1` is returned.

If `str` is null, a `NullPointerException` is thrown.

20.12.31 `public String substring(int beginIndex)`
throws `IndexOutOfBoundsException`

The result is a newly created `String` object that represents a subsequence of the character sequence represented by this `String` object; this subsequence begins with the character at position `beginIndex` and extends to the end of the character sequence.

If `beginIndex` is negative or larger than the length of this `String` object, then an `IndexOutOfBoundsException` is thrown.

Examples:

`"unhappy".substring(2)` returns `"happy"`

`"Harbison".substring(3)` returns `"bison"`

`"emptiness".substring(9)` returns `""` (an empty string)

20.12.32 `public String substring(int beginIndex, int endIndex)`
throws `IndexOutOfBoundsException`

The result is a newly created `String` object that represents a subsequence of the character sequence represented by this `String` object; this subsequence begins with the character at position `beginIndex` and ends with the character at position `endIndex-1`. Thus, the length of the subsequence is `endIndex-beginIndex`.

If `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`, then this method throws an `IndexOutOfBoundsException`.

Examples:

`"hamburger".substring(4, 8)` returns `"urge"`

`"smiles".substring(1, 5)` returns `"mile"`

20.12.33 `public String concat(String str)`
throws `NullPointerException`

If the length of the argument string is zero, then a reference to this `String` object is returned. Otherwise, a new `String` object is created, representing a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

`"cares".concat("s")` returns `"caress"`

`"to".concat("get").concat("her")` returns `"together"`

If `str` is null, a `NullPointerException` is thrown.

20.12.34 `public String replace(char oldChar, char newChar)`

If the character `oldChar` does not occur in the character sequence represented by this `String` object, then a reference to this `String` object is returned. Otherwise, a new `String` object is created that represents a character sequence identical to the character sequence represented by this `String` object, except that every occurrence of `oldChar` is replaced by an occurrence of `newChar`.

Examples:

```
"mesquite in your cellar".replace('e', 'o')
                                returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
                                returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
                                returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```

20.12.35 `public String toLowerCase()`

If this `String` object does not contain any character that is mapped to a different character by the method `Character.toLowerCase` ([§20.5.20](#)), then a reference to this `String` object is returned. Otherwise, this method creates a new `String` object that represents a character sequence identical in length to the character sequence represented by this `String` object, with every character equal to the result of applying the method `Character.toLowerCase` to the corresponding character of this `String` object.

Examples:

```
"French Fries".toLowerCase() returns "french fries"
"{ewc msdn cd, EWGraphic, LNG12x 7 /a "langref.BMP"}{ewc msdn cd, EWGraphic,
LNG12x 8 /a "langref.BMP"}{ewc msdn cd, EWGraphic, LNG12x 9 /a "langref.BMP"}
{ewc msdn cd, EWGraphic, LNG12x 10 /a "langref.BMP"}{ewc msdn cd, EWGraphic,
LNG12x 11 /a "langref.BMP"}".toLowerCase() returns "{ewc msdn cd, EWGraphic,
LNG12x 12 /a "langref.BMP"}{ewc msdn cd, EWGraphic, LNG12x 13 /a
"langref.BMP"}{ewc msdn cd, EWGraphic, LNG12x 14 /a "langref.BMP"}{ewc
msdn cd, EWGraphic, LNG12x 15 /a "langref.BMP"}{ewc msdn cd, EWGraphic,
LNG12x 16 /a "langref.BMP"}"
```

20.12.36 `public String toUpperCase()`

If this `String` object does not contain any character that is mapped to a different character by the method `Character.toUpperCase` ([§20.5.21](#)), then a reference to this `String` object is returned. Otherwise, this method creates a new `String` object representing a character sequence identical in length to the character sequence represented by this `String` object and with every character equal to the result of applying the method `Character.toUpperCase` to the corresponding character of this `String` object.

Examples:

```
"Fahrvergnngen".toUpperCase() returns "FAHRVERGNNGEN"
"Visit Ljubinja!".toUpperCase() returns "VISIT LJUBINJE!"
```


20.12.37 `public String trim()`

If this `String` object represents an empty character sequence, or the first and last characters of character sequence represented by this `String` object both have codes greater than `\u0020` (the space character), then a reference to this `String` object is returned.

Otherwise, if there is no character with a code greater than `\u0020` in the string, then a new `String` object representing an empty string is created and returned.

Otherwise, let k be the index of the first character in the string whose code is greater than `\u0020`, and let m be the index of the last character in the string whose code is greater than `\u0020`. A new `String` object is created, representing the substring of this string that begins with the character at index k and ends with the character at index m —that is, the result of `this.substring(k, m+1)`.

This method may be used to trim whitespace ([§20.5.19](#)) from the beginning and end of a string; in fact, it trims all ASCII control characters as well.

20.12.38 `public static String valueOf(Object obj)`

If the argument is `null`, then a string equal to `"null"` is returned. Otherwise, the value of `obj.toString()` is returned. See the `toString` method ([§20.1.2](#)).

20.12.39 `public static String valueOf(char[] data)`
throws `NullPointerException`

A string is created and returned. The string represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

20.12.40 `public static String valueOf(char[] data,`
 `int offset, int count)`
throws `NullPointerException,`
 `IndexOutOfBoundsException`

A string is created and returned. The string represents the sequence of characters currently contained in a subarray of the character array argument. The `offset` argument is the index of the first character of the subarray and the `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

If `data` is `null`, then a `NullPointerException` is thrown.

If `offset` is negative, or `count` is negative, or `offset+count` is larger than `data.length`, then an `IndexOutOfBoundsException` is thrown.

20.12.41 `public static String valueOf(boolean b)`

A string representation of `b` is returned.

If the argument is `true`, the string `"true"` is returned.

If the argument is `false`, the string `"false"` is returned.

20.12.42 `public static String valueOf(char c)`

A string is created and returned. The string contains one character, equal to `c`.

20.12.43 `public static String valueOf(int i)`

A string is created and returned. The string is computed exactly as if by the method `Integer.toString` of one argument [\(§20.7.12\)](#).

20.12.44 `public static String valueOf(long l)`

A string is created and returned. The string is computed exactly as if by the method `Long.toString` of one argument [\(§20.8.12\)](#).

20.12.45 `public static String valueOf(float f)`

A string is created and returned. The string is computed exactly as if by the method `Float.toString` of one argument [\(§20.9.16\)](#).

20.12.46 `public static String valueOf(double d)`

A string is created and returned. The string is computed exactly as if by the method `Double.toString` of one argument [\(§20.10.15\)](#).

20.12.47 `public String intern()`

A pool of strings, initially empty, is maintained privately by the class `String`.

When the `intern` method is invoked, if the pool already contains a string equal to this `String` object as determined by the `equals` method [\(§20.12.9\)](#), then the string from the pool is returned. Otherwise, this `String` object is added to the pool and a reference to this `String` object is returned.

It follows that for any two strings `s` and `t`, `s.intern() == t.intern()` is true if and only if `s.equals(t)` is true.

All literal strings and string-valued constant expressions are interned [\(§3.10.5\)](#).

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

20.13 The Class java.lang.StringBuffer

A string buffer is like a `String` ([§20.12](#)), but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

```
public class StringBuffer {
    public StringBuffer();
    public StringBuffer(int length)
        throws NegativeArraySizeException;
    public StringBuffer(String str);
    public String toString();
    public int length();
    public void setLength(int newLength)
        throws IndexOutOfBoundsException;
    public int capacity();
    public void ensureCapacity(int minimumCapacity);
    public char charAt(int index)
        throws IndexOutOfBoundsException;
    public void setCharAt(int index, char ch)
        throws IndexOutOfBoundsException;
    public void getChars(int srcBegin, int srcEnd,
        char[] dst, int dstBegin)
        throws NullPointerException, IndexOutOfBoundsException;
    public StringBuffer append(Object obj);
    public StringBuffer append(String str);
    public StringBuffer append(char[] str)
        throws NullPointerException;
    public StringBuffer append(char[] str, int offset, int len)
        throws NullPointerException, IndexOutOfBoundsException;
    public StringBuffer append(boolean b);
    public StringBuffer append(char c);
    public StringBuffer append(int i);
    public StringBuffer append(long l);
    public StringBuffer append(float f);
    public StringBuffer append(double d);
    public StringBuffer insert(int offset, Object obj)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, String str)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, char[] str)
        throws NullPointerException, IndexOutOfBoundsException;
    public StringBuffer insert(int offset, boolean b)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, char c)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, int i)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, long l)
        throws IndexOutOfBoundsException;
    public StringBuffer insert(int offset, float f)
        throws IndexOutOfBoundsException;
```



```

    public StringBuffer insert(int offset, double d)
        throws IndexOutOfBoundsException;
    public StringBuffer reverse();
}

```

A string buffer has a *capacity*. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to create a new internal buffer array.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary so that all the operations on any particular instance behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads involved.

String buffers can be used by a compiler to implement the binary string concatenation operator + ([§15.17.1](#)). For example, suppose `k` has type `int` and `a` has type `Object`. Then the expression:

```
k + "/" + a
```

can be compiled as if it were the expression:

```

new StringBuffer().append(k).append("/") .
    append(a).toString()

```

which creates a new string buffer (initially empty), appends the string representation of each operand to the string buffer in turn, and then converts the contents of the string buffer to a string. Overall, this avoids creating many temporary strings.

The principal operations on a `StringBuffer` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then adds the characters of that string to the contents of the string buffer. The `append` method always adds these characters at the end of the buffer; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string buffer object whose current contents are the characters "start", then the method call `z.append("le")` would alter the string buffer to contain the characters "startle", but `z.insert(4, "le")` would alter the string buffer to contain the characters "starlet".

In general, if `sb` refers to an instance of a `StringBuffer`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`.

20.13.1 `public StringBuffer()`

This constructor initializes a newly created `StringBuffer` object so that it initially represents an empty character sequence and has capacity 16.

20.13.2 `public StringBuffer(int length)` `throws NegativeArraySizeException`

This constructor initializes a newly created `StringBuffer` object so that it initially represents an empty character sequence, but has the capacity specified by the argument.

If the argument is negative, a `NegativeArraySizeException` is thrown.

20.13.3 `public StringBuffer(String str)`

This constructor initializes a newly created `StringBuffer` object so that it represents the same sequence of characters as the argument; in other words, the initial contents of the string buffer is a copy of the argument string. The initial capacity of the string buffer is 16 plus the length of the argument string.

20.13.4 `public String toString()`

A new `String` object is created and initialized to contain the character sequence currently represented by the string buffer; the new `String` is then returned. Any subsequent changes to the string buffer do not affect the contents of the returned string.

Implementation advice: This method can be coded so as to create a new `String` object without allocating new memory to hold a copy of the character sequence. Instead, the string can share the memory used by the string buffer. Any subsequent operation that alters the content or capacity of the string buffer must then make a copy of the internal buffer at that time. This strategy is effective for reducing the amount of memory allocated by a string concatenation operation ([§15.17.1](#)) when it is implemented using a string buffer.

Overrides the `toString` method of `Object` ([§20.1.2](#)).

20.13.5 `public int length()`

This method returns the length of the sequence of characters currently represented by this `StringBuffer` object.

20.13.6 `public int capacity()`

The current capacity of this `StringBuffer` object is returned.

20.13.7 `public void ensureCapacity(int minimumCapacity)`

If the current capacity of this `StringBuffer` object is less than the argument, then a new internal buffer is created with greater capacity. The new capacity will be the larger of:

- the `minimumCapacity` argument
- twice the old capacity, plus 2

If the `minimumCapacity` argument is nonpositive, this method takes no action and simply returns.

20.13.8 `public void setLength(int newLength)`
throws `IndexOutOfBoundsException`

This string buffer is altered to represent a new character sequence whose length is specified by the argument. For every nonnegative index k less than `newLength`, the character at index k in the new character sequence is the same as the character at index k in the old sequence if k is less than the length of the old character sequence; otherwise, it is the null character `'\u0000'`. This method also calls the `ensureCapacity` method ([§20.13.7](#)) with argument `newLength`.

If the argument is negative, an `IndexOutOfBoundsException` is thrown.

20.13.9 `public char charAt(int index)`
throws `IndexOutOfBoundsException`

The specified character of the sequence currently represented by the string buffer, as indicated by the `index` argument, is returned. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the `index` argument is negative or not less than the current length ([§20.13.5](#)) of the string buffer, an `IndexOutOfBoundsException` is thrown.

20.13.10 `public void setCharAt(int index, char ch)`
throws `IndexOutOfBoundsException`

The string buffer is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character `ch` at position `index`.

If the `index` argument is negative or not less than the current length ([§20.13.5](#)) of the string buffer, an `IndexOutOfBoundsException` is thrown.

20.13.11 `public void getChars(int srcBegin, int srcEnd,`
 `char[] dst, int dstBegin)`
throws `NullPointerException`,
 `IndexOutOfBoundsException`

Characters are copied from this `StringBuffer` object into the destination array `dst`. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus, the total number of characters to be copied is `srcEnd-srcBegin`). The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index `dstBegin+(srcEnd-srcBegin)-1`.

If `dst` is null, then a `NullPointerException` is thrown.

Otherwise, if any of the following is true, an `IndexOutOfBoundsException` is thrown and the destination is not modified:

- The `srcBegin` argument is negative.
- The `srcBegin` argument is greater than the `srcEnd` argument.
- `srcEnd` is greater than `this.length()`, the current length of this string buffer.
- `dstBegin+srcEnd-srcBegin` is greater than `dst.length`.

20.13.12 `public StringBuffer append(Object obj)`

The argument is converted to a string as if by the method `String.valueOf` ([§20.12.38](#)) and the characters of that string are then appended ([§20.13.13](#)) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.13 `public StringBuffer append(String str)`

The characters of the `String` argument are appended, in order, to the contents of this string buffer,

increasing the length of this string buffer by the length of the argument. If `str` is `null`, then the four characters "null" are appended to this string buffer. The method `ensureCapacity` ([§20.13.7](#)) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

Let n be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `append` method. Then the character at index k in the new character sequence is equal to the character at index k in the old character sequence, if k is less than n ; otherwise, it is equal to the character at index $k-n$ in the argument `str`.

20.13.14 `public StringBuffer append(char[] str)`
throws `NullPointerException`

The characters of the array argument are appended, in order, to the contents of this string buffer, increasing the length of this string buffer by the length of the argument. The method `ensureCapacity` ([§20.13.7](#)) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf` ([§20.12.39](#)) and the characters of that string were then appended ([§20.13.13](#)) to this `StringBuffer` object.

20.13.15 `public StringBuffer append(char[] str,`
 `int offset, int len)`
throws `NullPointerException`,
 `IndexOutOfBoundsException`

Characters of the character array `str`, starting at index `offset`, are appended, in order, to the contents of this string buffer, increasing the length of this string buffer by `len`. The method `ensureCapacity` ([§20.13.7](#)) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

The overall effect is exactly as if the arguments were converted to a string by the method `String.valueOf` of three arguments ([§20.12.40](#)) and the characters of that string were then appended ([§20.13.13](#)) to this `StringBuffer` object.

20.13.16 `public StringBuffer append(boolean b)`

The argument is converted to a string as if by the method `String.valueOf` ([§20.12.41](#)) and the characters of that string are then appended ([§20.13.13](#)) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.17 `public StringBuffer append(char c)`

The argument is appended to the contents of this string buffer, increasing the length of this string buffer by 1. The method `ensureCapacity` ([§20.13.7](#)) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf` ([§20.12.42](#)) and the character in that string were then appended ([§20.13.13](#)) to this `StringBuffer` object.

20.13.18 `public StringBuffer append(int i)`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.43) and the characters of that string are then appended (§20.13.13) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.19 `public StringBuffer append(long l)`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.44) and the characters of that string are then appended (§20.13.13) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.20 `public StringBuffer append(float f)`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.45) and the characters of that string are then appended (§20.13.13) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.21 `public StringBuffer append(double d)`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.46) and the characters of that string are then appended (§20.13.13) to this `StringBuffer` object. A reference to this `StringBuffer` object is returned.

20.13.22 `public StringBuffer insert(int offset, Object obj)`
 throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.38) and the characters of that string are then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.23 `public StringBuffer insert(int offset, String str)`
 throws `IndexOutOfBoundsException`

The characters of the `String` argument are inserted, in order, into the string buffer at the position indicated by `offset`, moving up any characters originally above that position and increasing the length of the string buffer by the length of the argument. If `str` is `null`, then the four characters "null" are inserted into this string buffer. The method `ensureCapacity` (§20.13.7) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

Let n be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `append` method. Then the character at index k in the new character sequence is equal to:

- the character at index k in the old character sequence, if k is less than `offset`
- the character at index $k - \text{offset}$ in the argument `str`, if k is not less than `offset` but is less than `offset + str.length()`
- the character at index $k - \text{str.length}()$ in the old character sequence, if k is not less than `offset + str.length()`

20.13.24 `public StringBuffer insert(int offset, char[] str)`
throws `NullPointerException`,
 `IndexOutOfBoundsException`

The characters of the array argument, taken in order, are inserted into this string buffer, increasing the length of the string buffer by the length of the argument. The method `ensureCapacity` (§20.13.7) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf` (§20.12.39) and the characters of that string were then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`.

Note that while the `StringBuffer` class provides an `append` method that takes an `offset`, a character array, and two other arguments (§20.13.15), it does not currently provide an `insert` method that takes an `offset`, a character array, and two other arguments.

20.13.25 `public StringBuffer insert(int offset, boolean b)`
 throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.41) and the characters of that string are then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.26 `public StringBuffer insert(int offset, char c)`
 throws `IndexOutOfBoundsException`

The argument is inserted into the contents of this string buffer at the position indicated by `offset`, increasing the length of this string buffer by 1. The method `ensureCapacity` (§20.13.7) is first called with this new string buffer length as its argument. A reference to this `StringBuffer` object is returned.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf` (§20.12.42) and the character in that string were then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`.

20.13.27 `public StringBuffer insert(int offset, int i)`
 throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.43) and the characters of that string are then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.28 `public StringBuffer insert(int offset, long l)`
 throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.44) and the characters of that string are inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.29 `public StringBuffer insert(int offset, float f)`

throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.45) and the characters of that string are then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.30 `public StringBuffer insert(int offset, double d)`
 throws `IndexOutOfBoundsException`

The argument is converted to a string as if by the method `String.valueOf` (§20.12.46) and the characters of that string are then inserted (§20.13.23) into this `StringBuffer` object at the position indicated by `offset`. A reference to this `StringBuffer` object is returned.

20.13.31 `public StringBuffer reverse()`

The character sequence contained in this `StringBuffer` object is replaced by the reverse of that sequence. A reference to this `StringBuffer` object is returned.

Let n be the length of the old character sequence, the one contained in the string buffer just prior to execution of the `reverse` method. Then the character at index k in the new character sequence is equal to the character at index $n-k-1$ in the old character sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.14 The Class `java.lang.ClassLoader`

A class loader is an object that is responsible for loading classes. Given the name of a class, it should attempt to locate or generate data that constitutes a definition for the class. A typical strategy is to transform the name into a file name and then read a "class file" of that name from a file system.

Every `Class` object contains a reference to the `ClassLoader` that defined it ([§20.3.7](#)). Whenever executable Java code needs to use a class that has not yet been loaded, the `loadClass` method is invoked for the class loader of the class containing the code in question.

Class objects for array classes are not created by class loaders, but are created automatically as required by the Java runtime. The class loader for an array class, as returned by the `getClassLoader` method of class `Class` ([§20.3.7](#)), is the same as the class loader for its element type; if the element type is a primitive type, then the array class has no class loader.

Class loaders may typically be used by security managers ([§20.17](#)) to indicate security domains: two classes may be considered to be "friendly" or "related" to each other only if they were defined by the same class loader.

```
public abstract class ClassLoader {
    protected ClassLoader()

        throws SecurityException;
    protected abstract Class loadClass(String name,
        boolean resolve)

        throws ClassNotFoundException;
    protected final Class defineClass(byte data[],
        int offset, int length)

        throws NullPointerException, IndexOutOfBoundsException,
            ClassFormatError;
    protected final void resolveClass(Class c);
    protected final Class findSystemClass(String name)

        throws ClassNotFoundException;
}
```

20.14.1 `protected ClassLoader() throws SecurityException`

This constructor is invoked for every newly created class loader. Because the class `ClassLoader` is abstract, it is not possible to create a new instance of the class `ClassLoader` itself; however, every constructor for a subclass of `ClassLoader` necessarily invokes this constructor, explicitly or implicitly, directly or indirectly.

All this constructor does is to enforce a security check: if there is a security manager, its `checkCreateClassLoader` method ([§20.17.10](#)) is called.

20.14.2 `protected abstract Class loadClass(String name, boolean link) throws ClassNotFoundException`

Every subclass of `ClassLoader` that is not itself abstract must provide an implementation of the

method `loadClass`.

The general contract of `loadClass` is that, given the name of a class, it either returns the `Class` object for the class or throws a `ClassNotFoundException`.

If a `Class` object is to be returned and `link` is `true`, then the `Class` object should be linked ([§12.3](#), [§20.14.4](#)) before it is returned.

In most cases, it is wise for a subclass of `ClassLoader` ([§20.14](#)) to implement the `loadClass` method as a `synchronized` method.

```
20.14.3      protected final Class defineClass(byte data[],
          int offset, int length)
throws NullPointerException,      IndexOutOfBoundsException,
ClassFormatError
```

This method may be used by a class loader to define a new class.

The bytes in the array `data` in positions `offset` through `offset+length-1` should have the format of a valid class file as defined by the *Java Virtual Machine Specification*.

If data is null, then a `NullPointerException` is thrown.

An `IndexOutOfBoundsException` is thrown if any of the following are true:

- `offset` is negative
- `length` is negative
- `offset+length` is greater than `data.length`

If the indicated bytes of `data` do not constitute a valid class definition, then a `ClassFormatError` is thrown. Otherwise, this method creates and returns a `Class` object as described by the data bytes

```
20.14.4    protected final void resolveClass(Class c)
            throws NullPointerException
```

This (misleadingly named) method may be used by a class loader to link ([§12.3](#), [§20.14.4](#)) a class.

If `c` is null, then a `NullPointerException` is thrown.

If the `Class` object `c` has already been linked, then this method simply returns.

Otherwise, the class is linked as described in §12.3.

```
20.14.5      protected final Class findSystemClass(String name)
throws ClassNotFoundException
```

This method may be used by a class loader to locate a class that has no class loader. This includes built-in classes such as `java.lang.Object`, as well as classes that the host implementation may keep in, for example, a local file system.

Given the `name` of a class, this method, like the `loadClass` method, either returns the `Class` object for the class or throws a `ClassNotFoundException`.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.15 The Class java.lang.Process

The method `exec` ([§20.16.3](#)) of class `Runtime` returns a reference to a `Process` object. The class `Process` provides methods for performing input from the process, performing output to the process, waiting for the process to complete, checking the exit status of the process, and destroying (killing) the process.

Dropping the last reference to a `Process` instance, thus allowing the `Process` object to be reclaimed, does *not* automatically kill the associated process.

There is no requirement that a process represented by a `Process` object execute asynchronously or concurrently with respect to the Java process that owns the `Process` object.

```
public abstract class Process {
    public abstract OutputStream getOutputStream();
    public abstract InputStream getInputStream();
    public abstract InputStream getErrorStream();
    public abstract int waitFor()

        throws InterruptedException;
    public abstract int exitValue()

        throws IllegalStateException;
    public abstract void destroy();
}
```

20.15.1 `public abstract OutputStream getOutputStream()`

This method returns an `OutputStream`. Output to the stream is piped into the standard input stream of the process represented by this `Process` object.

Implementation note: It is a good idea for the output stream to be buffered.

20.15.2 `public abstract InputStream getInputStream()`

This method returns an `InputStream`. The stream obtains data piped from the standard output stream of the process represented by this `Process` object.

Implementation note: It is a good idea for the input stream to be buffered.

20.15.3 `public abstract InputStream getErrorStream()`

This method returns an `InputStream`. The stream obtains data piped from the error output stream of the process represented by this `Process` object.

Implementation note: It is a good idea for the input stream to be buffered.

20.15.4 `public abstract int waitFor()` `throws InterruptedException`

This method causes the current thread to wait, if necessary, until the process represented by this

`Process` object has terminated. Then the exit value of the process is returned. By convention, the value 0 indicates normal termination.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then the wait is ended and an `InterruptedException` is thrown.

20.15.5 `public abstract int exitValue()`
throws `IllegalThreadStateException`

If the process represented by this `Process` object has not yet terminated, then an `IllegalThreadStateException` is thrown. Otherwise, the exit value of the process is returned. By convention, the value 0 indicates normal termination.

20.15.6 `public abstract void destroy()`

The process represented by this `Process` object is forcibly terminated.

It was my hint to speak--such was the process.

--William Shakespeare, *Othello*, Act I, scene iii

{`ewl msdncd.dll, ewcright, /c"Microsoft"`}

20.16 The Class java.lang.Runtime

```
public class Runtime {
    public static Runtime getRuntime();
    public void exit(int status);
    public Process exec(String command)

        throws IOException, SecurityException;
    public Process exec(String command, String envp[])

        throws IOException, SecurityException;
    public Process exec(String cmdarray[])

        throws IOException, SecurityException;
    public Process exec(String cmdarray[], String envp[])

        throws IOException, SecurityException;
    public long totalMemory();
    public long freeMemory();
    public void gc();
    public void runFinalization();
    public void traceInstructions(boolean on);
    public void traceMethodCalls(boolean on);
    public void load(String filename)

        throws SecurityException, UnsatisfiedLinkError;
    public void loadLibrary(String libname)

        throws SecurityException, UnsatisfiedLinkError;
    public InputStream getLocalizedInputStream(InputStream in);
    public OutputStream

        getLocalizedOutputStream(OutputStream out);
}
```

20.16.1 `public static Runtime getRuntime()`

This method returns the current `Runtime` object. Most of the methods of class `Runtime` are instance methods and must be invoked with respect to the current runtime object.

20.16.2 `public void exit(int status)` `throws SecurityException`

First, if there is a security manager, its `checkExit` method ([§20.17.13](#)) is called with the `status` value as its argument.

This method terminates the currently running Java Virtual Machine. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

This method never returns normally.

See also the method `exit` ([§20.18.11](#)) of class `System`, which is the conventional and convenient means of invoking this method.

20.16.3 `public Process exec(String command)`
throws `IOException`, `SecurityException`,

`IndexOutOfBoundsException`

The `command` argument is parsed into tokens and then executed as a command in a separate process. The token parsing is done by a `StringTokenizer` ([§21.10](#)) created by the call:

```
new StringTokenizer(command)
```

with no further modification of the character categories.

This method behaves exactly as if it performs the call:

```
exec(command, null)
```

See [§20.16.4](#).

20.16.4 `public Process exec(String command, String envp[])`
throws `IOException`, `SecurityException`,
 `IndexOutOfBoundsException`

The `command` argument is parsed into tokens and then executed as a command in a separate process with an environment specified by `envp`. The token parsing is done by a `StringTokenizer` ([§21.10](#)) created by the call:

```
new StringTokenizer(command)
```

with no further modification of the character categories.

This method breaks the `command` string into tokens and creates a new array `cmdarray` containing the tokens in the order that they were produced by the string tokenizer; it then behaves exactly as if it performs the call:

```
exec(cmdarray, envp)
```

See [§20.16.6](#).

20.16.5 `public Process exec(String cmdarray[])`
throws `IOException`, `SecurityException`,
 `NullPointerException`, `IndexOutOfBoundsException`

The command specified by the tokens in `cmdarray` is executed as a command in a separate process.

This method behaves exactly as if it performs the call:

```
exec(cmdarray, null)
```


See [§20.16.6](#).

20.16.6 `public Process exec(String cmdarray[], String envp[])`
throws `IOException`, `SecurityException`,
 `NullPointerException`, `IndexOutOfBoundsException`

First, if there is a security manager, its `checkExec` method ([§20.17.14](#)) is called with the first component of the array `cmdarray` as its argument.

If `cmdarray` is `null`, a `NullPointerException` is thrown. If `cmdarray` is an empty array (has length 0), an `IndexOutOfBoundsException` is thrown.

Given an array of strings `cmdarray`, representing the tokens of a command line, and an array of strings `envp`, representing an "environment" that defines system properties, this method creates a new process in which to execute the specified command and returns a `Process` object ([§20.15](#)) representing the new process.

20.16.7 `public long totalMemory()`

The total amount of memory currently available for current and future created objects, measured in bytes, is returned. The value returned by this method may vary over time, depending on the host environment.

Note that the amount of memory required to hold an object of any given type may be implementation-dependent.

20.16.8 `public long freeMemory()`

An approximation to the total amount of memory currently available for future created objects, measured in bytes, is returned. This value is always less than the current value returned by the `totalMemory` method. Calling the `gc` method may increase the value returned by `freeMemory`.

20.16.9 `public void gc()`

Calling this method suggests that the Java Virtual Machine expend effort toward recycling discarded objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to recycle all discarded objects. (The name `gc` stands for "garbage collector.")

The Java runtime system will perform this recycling process automatically as needed, in a separate thread, if the `gc` method is not invoked explicitly.

See also the method `gc` ([§20.18.12](#)) of class `System`, which is the conventional and convenient means of invoking this method.

20.16.10 `public void runFinalization()`

Calling this method suggests that the Java Virtual Machine expend effort toward running the `finalize` methods of objects that have been found to be discarded but whose `finalize` methods have not yet been run. When control returns from the method call, the Java Virtual Machine has made a best effort to complete all outstanding finalizations.

The Java runtime system will perform the finalization process automatically as needed, in a separate thread, if the `runFinalization` method is not invoked explicitly.

See also the method `runFinalization` ([§20.18.13](#)) of class `System`, which is the conventional and convenient means of invoking this method.

20.16.11 `public void traceInstructions(boolean on)`

Calling this method with argument `true` suggests that the Java Virtual Machine emit debugging information for every instruction it executes. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment.

Calling this method with argument `false` suggests that the Java Virtual Machine cease emitting per-instruction debugging information.

20.16.12 `public void traceMethodCalls(boolean on)`

Calling this method with argument `true` suggests that the Java Virtual Machine emit debugging information for every method call it executes. The format of this information, and the file or other output stream to which it is emitted, depends on the host environment.

Calling this method with argument `false` suggests that the Java Virtual Machine cease emitting per-call debugging information.

20.16.13 `public void loadLibrary(String libname)`

First, if there is a security manager, its `checkLink` method ([§20.17.17](#)) is called with the `libname` as its argument.

A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent.

See also the method `loadLibrary` ([§20.18.15](#)) of class `System`, which is the conventional and convenient means of invoking this method. If native methods are to be used in the implementation of a class, a standard strategy is to put the native code in a library file (call it `LibFile`) and then to put a static initializer:

```
static { System.loadLibrary("LibFile"); }
```

within the class declaration. When the class is loaded and initialized ([§12.4](#)), the necessary native code implementation for the `native` methods will then be loaded as well.

20.16.14 `public void load(String filename)`

First, if there is a security manager, its `checkLink` method ([§20.17.17](#)) is called with the `filename` as its argument.

This is similar to the method `loadLibrary` ([§20.16.13](#)), but accepts a general file name as an argument rather than just a library name, allowing any file of native code to be loaded.

See also the method `load` ([§20.18.14](#)) of class `System`, which is the conventional and convenient means of invoking this method.

20.16.15 `public InputStream
getLocalizedInputStream(InputStream in)`

This method takes an `InputStream` ([§22.3](#)) and returns an `InputStream` equivalent to the argument in all respects except that it is localized: as data is read from the stream, it is automatically converted from the local format to Unicode. If the argument is already a localized stream, then it will be returned as the result.

20.16.16 `public OutputStream
getLocalizedOutputStream(OutputStream out)`

This method takes an `OutputStream` ([§22.15](#)) and returns an `OutputStream` equivalent to the argument in all respects except that it is localized: as data is written to the stream, it is automatically converted from Unicode to the local format. If the argument is already a localized stream, then it will be returned as the result.

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

20.17 The Class java.lang.SecurityManager

```
public abstract class SecurityManager {
    protected boolean inCheck;
    protected SecurityManager()

        throws SecurityException;
    protected Class[] getClassContext();
    protected int classDepth(String name);
    protected boolean inClass(String name);
    protected ClassLoader currentClassLoader();
    protected int classLoaderDepth();
    protected boolean inClassLoader();
    public boolean getInCheck();
    public void checkCreateClassLoader()

        throws SecurityException;
    public void checkAccess(Thread t)

        throws SecurityException;
    public void checkAccess(ThreadGroup g)

        throws SecurityException;
    public void checkExit(int status)

        throws SecurityException;
    public void checkExec(String cmd)

        throws SecurityException;
    public void checkPropertiesAccess()

        throws SecurityException;
    public void checkPropertyAccess(String key)

        throws SecurityException;
    public void checkLink(String libname)

        throws SecurityException;
    public void checkRead(int fd)

        throws SecurityException;
    public void checkRead(String file)

        throws SecurityException;
    public void checkWrite(int fd)

        throws SecurityException;
    public void checkWrite(String file)

        throws SecurityException;
    public void checkDelete(String file)

        throws SecurityException;
    public void checkConnect(String host, int port)
```



```

        throws SecurityException;
public void checkListen(int port)

        throws SecurityException;
public void checkAccept(String host, int port)

        throws SecurityException;
public void checkSetFactory()

        throws SecurityException;
public boolean checkTopLevelWindow()

        throws SecurityException;
public void checkPackageAccess(String packageName)

        throws SecurityException;
public void checkPackageDefinition(String packageName)

        throws SecurityException;
}

```

A running Java program may have a security manager, which is an instance of class `SecurityManager`. The current security manager is the one returned by the method invocation `System.getSecurityManager()` ([§20.18.4](#)).

The `SecurityManager` class contains a large number of methods whose names begin with "check". They are called by various methods throughout the Java libraries before those methods perform certain sensitive operations. The invocation of such a check method typically looks like this:

```

SecurityManager security = System.getSecurityManager();
if (security != null) {
    security.checkXXX(arguments);
}

```

The security manager is thereby given an opportunity to prevent completion of the operation by throwing an exception. The usual convention is that a security manager checking routine simply returns if the operation is permitted, or throws a `SecurityException` if the operation is not permitted. In one case, namely `checkTopLevelWindow` ([§20.17.27](#)), the checking routine must return a `boolean` value to indicate one of two levels of permission.

20.17.1 `protected boolean inCheck = false;`

By convention, this field should be assigned the value `true` whenever a security check is in progress. This matters when one of the checking routines needs to call outside code to do its work. Outside code can then use the method `getInCheck` ([§20.17.9](#)) to test the status of this flag.

20.17.2 `protected SecurityManager()`
`throws SecurityException`

This constructor checks to see whether a security manager has already been installed ([§20.18.5](#)); if so, creation of another security manager is not permitted, and so a `SecurityException` is thrown.

20.17.3 `protected Class[] getClassContext()`

This utility method for security managers scans the execution stack for the current thread and returns an array with one component for each stack frame. The component at position 0 corresponds to the top of the stack. If a component is a `Class` object, then the corresponding stack frame is for an invocation of a method of the class represented by that `Class` object.

20.17.4 `protected int classDepth(String name)`

This utility method for security managers searches the execution stack for the current thread to find the most recently invoked method whose execution has not yet completed and whose class has `name` as its fully qualified name. If such a method is found, its distance from the top of the stack is returned as a nonnegative integer; otherwise, `-1` is returned.

20.17.5 `protected boolean inClass(String name)`

This utility method for security managers searches the execution stack for the current thread to find the most recently invoked method whose execution has not yet completed and whose class has `name` as its fully qualified name. If such a method is found, `true` is returned; otherwise, `false` is returned.

20.17.6 `protected ClassLoader currentClassLoader()`

This utility method for security managers searches the execution stack for the current thread to find the most recently invoked method whose execution has not yet completed and whose class was created by a class loader ([§20.14](#)). If such a method is found, a reference to the `ClassLoader` object for its class is returned; otherwise, `null` is returned.

20.17.7 `protected int classLoaderDepth()`

This utility method for security managers searches the execution stack for the current thread to find the most recently invoked method whose execution has not yet completed and whose class was created by a class loader ([§20.14](#)). If such a method is found, its distance from the top of the stack is returned as a nonnegative integer; otherwise, `-1` is returned.

20.17.8 `protected boolean inClassLoader()`

This utility method for security managers searches the execution stack for the current thread to find the most recently invoked method whose execution has not yet completed and whose class was created by a class loader ([§20.14](#)). If such a method is found, `true` is returned; otherwise `false` is returned.

20.17.9 `public boolean getInCheck()`

The value of the `inCheck` field ([§20.17.1](#)) is returned.

20.17.10 `public void checkCreateClassLoader()`
`throws SecurityException`

The general contract of this method is that it should throw a `SecurityException` if creation of a class loader is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by the constructor for class `ClassLoader` ([§20.14.1](#)).

The `checkCreateClassLoader` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a class loader creation operation is to be permitted with a security manager installed.

20.17.11 `public void checkAccess(Thread t)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if an operation that would modify the thread `t` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by method `checkAccess` ([§20.20.12](#)) of class `Thread`.

The `checkAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a thread modification operation is to be permitted with a security manager installed.

20.17.12 `public void checkAccess(ThreadGroup g)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if an operation that would modify the thread group `g` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by method `checkAccess` ([§20.21.4](#)) of class `ThreadGroup`.

The `checkAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a thread group modification operation is to be permitted with a security manager installed.

20.17.13 `public void checkExit(int status)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if an exit operation that would terminate the running Java Virtual Machine is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by method `exit` ([§20.16.2](#)) of class `Runtime`.

The `checkExit` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if the exit operation is to be permitted with a security manager installed.

20.17.14 `public void checkExec(String cmd)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if a command `exec` operation is not permitted. The argument `cmd` is the name of the command to be executed.

This method is invoked for the current security manager ([§20.18.4](#)) by method `exec` ([§20.16.6](#)) of

class Runtime.

The `checkExec` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a command `exec` operation is to be permitted with a security manager installed.

20.17.15 `public void checkPropertiesAccess()`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if getting or setting the system properties data structure is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by the methods `getProperties` ([§20.18.7](#)) and `setProperties` ([§20.18.8](#)) of class `System`.

The `checkPropertiesAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a properties access operation is to be permitted with a security manager installed.

20.17.16 `public void checkPropertyAccess(String key)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if getting the value of the system property named by the `key` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by the methods `getProperty` of one argument ([§20.18.9](#)) and `getProperty` of two arguments ([§20.18.10](#)) of class `System`.

The `checkPropertyAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if accessing the value of a system property is to be permitted with a security manager installed.

20.17.17 `public void checkLink(String libname)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if dynamic linking of the specified library code file is not permitted. The argument may be a simple library name or a complete file name.

This method is invoked for the current security manager ([§20.18.4](#)) by methods `load` ([§20.16.14](#)) and `loadLibrary` ([§20.16.13](#)) of class `Runtime`.

The `checkLink` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a dynamic code linking operation is to be permitted with a security manager installed.

20.17.18 `public void checkRead(int fd)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if creating an input stream using the specified file descriptor is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by one constructor for `java.io.FileInputStream` ([§22.4.3](#)).

The `checkRead` method defined by class `SecurityManager` always throws a

`SecurityException`. A subclass must override this method if creating an input stream from an existing file descriptor is to be permitted with a security manager installed.

20.17.19 `public void checkRead(String file)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if reading the specified file or directory, or examining associated file-system information, or testing for its existence, is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by two constructors for `java.io.FileInputStream` ([§22.4.1](#), [§22.4.2](#)); by two constructors for `java.io.RandomAccessFile` ([§22.23.1](#), [§22.23.2](#)); and by methods `exists` ([§22.24.16](#)), `canRead` ([§22.24.17](#)), `isFile` ([§22.24.19](#)), `isDirectory` ([§22.24.20](#)), `lastModified` ([§22.24.21](#)), `length` ([§22.24.22](#)), `list` with no arguments ([§22.24.25](#)), and `list` with one argument ([§22.24.26](#)) of the class `java.io.File`.

The `checkRead` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if read access to a file is to be permitted with a security manager installed.

20.17.20 `public void checkWrite(int fd)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if creating an output stream using the specified file descriptor is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by one constructor for `java.io.FileOutputStream` ([§22.16.3](#)).

The `checkWrite` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if creating an output stream from an existing file descriptor is to be permitted with a security manager installed.

20.17.21 `public void checkWrite(String file)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if writing, modifying, creating (for output), or renaming the specified file or directory is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by two constructors for `java.io.FileOutputStream` ([§22.16.1](#), [§22.16.2](#)); by two constructors for `java.io.RandomAccessFile` ([§22.23.1](#), [§22.23.2](#)); and by methods `canWrite` ([§22.24.18](#)), `mkdir` ([§22.24.23](#)), and `renameTo` ([§22.24.27](#)) of class `java.io.File`.

The `checkWrite` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if write access to a file is to be permitted with a security manager installed.

20.17.22 `public void checkDelete(String file)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if deleting the specified file is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by method `delete` ([§22.24.28](#)) of class `java.io.File`.

The `checkDelete` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a file deletion operation is to be permitted with a security manager installed.

20.17.23 `public void checkConnect(String host, int port)`
 throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if connecting to the indicated `port` of the indicated network `host` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by two constructors for class `java.net.Socket`, methods `send` and `receive` of class `java.net.DatagramSocket`, and methods `getByName` and `getAllByName` of class `java.net.InetAddress`. (These classes are not documented in this specification. See *The Java Application Programming Interface*.)

The `checkConnect` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a network connection is to be permitted with a security manager installed.

20.17.24 `public void checkListen(int port)`
 throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if listening to the specified local network `port` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by the constructor of one argument for class `java.net.DatagramSocket` and by the constructors for class `java.net.ServerSocket`. (These classes are not documented in this specification. See *The Java Application Programming Interface*.)

The `checkListen` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if listening to a local network port is to be permitted with a security manager installed.

20.17.25 `public void checkAccept(String host, int port)`
 throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if accepting a connection from the indicated `port` of the indicated network `host` is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by method `accept` of class `java.net.ServerSocket`. (This class is not documented in this specification. See *The Java Application Programming Interface*.)

The `checkAccept` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if accepting a network connection is to be permitted with a security manager installed.

20.17.26 `public void checkSetFactory()`
 throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if installing a

"factory" for a socket, server socket, URL, or URL connection is not permitted.

This method is invoked for the current security manager ([§20.18.4](#)) by:

```
method setSocketFactory of class java.net.ServerSocket
method setSocketImplFactory of class java.net.Socket
method setURLStreamHandlerFactory of class java.net.URL
method setContentHandlerFactory of class java.net.URLConnection
```

(These classes are not documented in this specification. See *The Java Application Programming Interface*.)

The `checkSetFactory` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if a factory installation operation is to be permitted with a security manager installed.

20.17.27 `public boolean checkTopLevelWindow()`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if creation of a top-level window is not permitted. If creation of a top-level window is permitted, then this method should return `false` if the window ought to bear a clear warning that it is a window for an executable applet. A returned value of `true` means that the security manager places no restriction on window creation.

This method is invoked for the current security manager ([§20.18.4](#)) by the constructors for class `java.awt.Window`. (This class is not documented in this specification. See *The Java Application Programming Interface*.)

The `checkTopLevelWindow` method defined by class `SecurityManager` always returns `false`. A subclass must override this method if a window creation operation is to be unrestricted or forbidden with a security manager installed.

20.17.28 `public void checkPackageAccess(String packageName)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if the current applet is not permitted to access the package named by the argument. This method is intended for use by Java-capable web browsers.

The `checkPackageAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if package access by an applet is to be permitted with a security manager installed.

20.17.29 `public void checkPackageDefinition(String packageName)`
throws `SecurityException`

The general contract of this method is that it should throw a `SecurityException` if the current applet is not permitted to define a class (or interface) in the package named by the argument. This method is intended for use by Java-capable web browsers.

The `checkPackageAccess` method defined by class `SecurityManager` always throws a `SecurityException`. A subclass must override this method if class definition by an applet is to be permitted with a security manager installed.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.18 The Class java.lang.System

The `System` class contains a number of useful class variables and class methods. It cannot be instantiated. Among the facilities provided by the `System` class are standard input, output, and error output streams; access to externally defined "properties"; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

```
public final class System {
    public static InputStream in;
    public static PrintStream out;
    public static PrintStream err;
    public static SecurityManager getSecurityManager();
    public static void setSecurityManager(SecurityManager s)
        throws SecurityException;
    public static long currentTimeMillis();
    public static Properties getProperties()
        throws SecurityException;
    public static void setProperties(Properties props)
        throws SecurityException;
    public static String getProperty(String key)
        throws SecurityException;
    public static String getProperty(String key, String default)
        throws SecurityException;
    public static void exit(int status) throws SecurityException;
    public static void gc();
    public static void runFinalization();
    public static void load(String filename)
        throws SecurityException, UnsatisfiedLinkError;
    public static void loadLibrary(String libname)
        throws SecurityException, UnsatisfiedLinkError;
    public static void arraycopy(Object src, int srcOffset,
        Object dst, int dstOffset, int length)

        throws NullPointerException,
        ArrayStoreException, IndexOutOfBoundsException;
}
```

20.18.1 `public static InputStream in;`

The initial value of this variable is a "standard" input stream, already open and ready to supply input data. Typically, this corresponds to keyboard input or another input source specified by the host environment or user. Note that this field is not `final`, so its value may be updated if necessary.

20.18.2 `public static PrintStream out;`

The initial value of this variable is a "standard" output stream, already open and ready to accept output data. Typically, this corresponds to display output or another output destination specified by the host environment or user. Note that this field is not `final`, so its value may be updated if necessary.

For simple Java applications, a typical way to write a line of output data is:


```
System.out.println(data)
```

See the `println` method of class `PrintStream` ([§22.22](#)).

20.18.3 `public static PrintStream err;`

The initial value of this variable is a "standard" error output stream, already open and ready to accept output data. Typically, this corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable `out`, has been redirected to a file or other destination that is typically not continuously monitored. Note that this field is not `final`, so its value may be updated if necessary.

20.18.4 `public static SecurityManager getSecurityManager()`

If a security manager has already been established for the currently running Java system, a reference to that security manager is returned. Otherwise, `null` is returned.

20.18.5 `public static void setSecurityManager(SecurityManager s)`
`throws SecurityException`

If a security manager has already been established for the currently running Java system, a `SecurityException` is thrown. Otherwise, the argument is established as the current security manager. If the argument is `null` and no security manager has been established, then no action is taken and the method simply returns normally.

20.18.6 `public static long currentTimeMillis()`

Returns the difference, measured in milliseconds, between the current time and the standard base time known as "the epoch," 00:00:00 GMT on January 1, 1970. See the description of the class `Date` ([§21.3](#)) for a discussion of slight discrepancies that may arise between "computer time" and UTC (Coordinated Universal Time).

20.18.7 `public static Properties getProperties()`
`throws SecurityException`

First, if there is a security manager, its `checkPropertiesAccess` method ([§20.17.15](#)) is called with no arguments.

The current set of system properties for use by the `getProperty` method is returned as a `Properties` object ([§21.6](#)). If there is no current set of system properties, a set of system properties is first created and initialized. This set of system properties always includes values for the following keys:

<i>Key</i>	<i>Description of associated value</i>
<code>java.version</code>	Java version number
<code>java.vendor</code>	Java-vendor-specific
<code>string</code>	

<code>java.vendor.url</code>	Java vendor URL
<code>java.home</code>	Java installation directory
<code>java.class.version</code>	Java class
<code>format version number</code>	
<code>java.class.path</code>	Java classpath
<code>os.name</code>	Operating system name
<code>os.arch</code>	Operating system
<code>architecture</code>	
<code>os.version</code>	Operating system version
<code>file.separator</code>	File separator (/ on
UNIX)	
<code>path.separator</code>	Path separator (: on
UNIX)	
<code>line.separator</code>	Line separator (\n on
UNIX)	
<code>user.name</code>	User account name
<code>user.home</code>	User home directory
<code>user.dir</code>	User's current working
<code>directory</code>	

Note that even if the security manager does not permit the `getProperties` operation, it may choose to permit the `getProperty` operation (§20.18.9).

20.18.8 `public static void setProperties(Properties props)`
throws `SecurityException`

First, if there is a security manager, its `checkPropertiesAccess` method (§20.17.15) is called with no arguments.

The argument becomes the current set of system properties for use by the `getProperty` method. See the class `Properties` (§21.6). If the argument is `null`, then the current set of system properties is forgotten.

20.18.9 `public static String getProperty(String key)`
throws `SecurityException`

First, if there is a security manager, its `checkPropertyAccess` method (§20.17.16) is called with the `key` as its argument.

If there is no current set of system properties, a set of system properties is first created and initialized in the same manner as for the `getProperties` method (§20.18.7).

The system property value associated with the specified `key` string is returned. If there is no property with that key, then `null` is returned.

20.18.10 `public static String getProperty(String key,`
 `String default)`
throws `SecurityException`

First, if there is a security manager, its `checkPropertyAccess` method (§20.17.16) is called with the `key` as its argument.

If there is no current set of system properties, a set of system properties is first created and initialized in the same manner as for the `getProperties` method (§20.18.7).

The system property value associated with the specified `key` string is returned. If there is no property with that key, then the argument `default` is returned.

20.18.11 `public static void exit(int status)`
throws `SecurityException`

This method terminates the currently running Java Virtual Machine. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

This method never returns normally.

The call `System.exit(n)` is effectively equivalent to the call:

```
Runtime.getRuntime().exit(n)
```

For a more complete description, see the `exit` method of class `Runtime` ([§20.16.2](#)).

20.18.12 `public static void gc()`

Calling this method suggests that the Java Virtual Machine expend effort toward recycling discarded objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to recycle all discarded objects.

The call `System.gc()` is effectively equivalent to the call:

```
Runtime.getRuntime().gc()
```

For a more complete description, see the `gc` method of class `Runtime` ([§20.16.9](#)).

20.18.13 `public static void runFinalization()`

Calling this method suggests that the Java Virtual Machine expend effort toward running the finalization methods of objects that have been found to be discarded but whose finalization methods have not yet been run. When control returns from the method call, the Java Virtual Machine has made a best effort to complete all outstanding finalizations.

The call `System.runFinalization()` is effectively equivalent to the call:

```
Runtime.getRuntime().runFinalization()
```

For a more complete description, see the `runFinalization` method of class `Runtime` ([§20.16.10](#)).

20.18.14 `public static void load(String filename)`
throws `SecurityException`, `UnsatisfiedLinkError`

This method loads a code file with the specified file name from the local file system.

The call `System.load(name)` is effectively equivalent to the call:


```
Runtime.getRuntime().load(name)
```

For a more complete description, see the `load` method of class `Runtime` ([§20.16.14](#)).

20.18.15 `public static void loadLibrary(String libname)`
 throws `SecurityException`, `UnsatisfiedLinkError`

This method loads a library code file with the specified library name from the local file system.

The call `System.loadLibrary(name)` is effectively equivalent to the call

```
Runtime.getRuntime().loadLibrary(name)
```

For a more complete description, see the `loadLibrary` method of class `Runtime` ([§20.16.13](#)).

20.18.16 `public static void arraycopy(Object src, int srcOffset,`
 `Object dst, int dstOffset, int length)`
 throws `NullPointerException`, `ArrayStoreException`,
 `IndexOutOfBoundsException`

A subsequence of array components is copied from the source array referenced by `src` to the destination array referenced by `dst`. The number of components copied is equal to the `length` argument. The components at the positions `srcOffset` through `srcOffset+length-1` in the source array are copied into the positions `dstOffset` through `dstOffset+length-1`, respectively, of the destination array.

If the `src` and `dst` arguments refer to the same array object, then copying is performed as if the components of the source array at positions `srcOffset` through `srcOffset+length-1` were first copied to a temporary array of length `length` and then the contents of the temporary array were copied into positions `dstOffset` through `dstOffset+length-1` of the destination array.

If `dst` is `null`, then a `NullPointerException` is thrown.

If `src` is `null`, then a `NullPointerException` is thrown and the destination array is not modified.

Otherwise, if any of the following is true, then an `ArrayStoreException` is thrown and the destination is not modified:

- The `src` argument refers to an object that is not an array.
- The `dst` argument refers to an object that is not an array.
- The `src` argument and `dst` argument refer to arrays whose component types are different primitive types.
- The `src` argument refers to an array of primitive component type and the `dst` argument refers to an array of reference component type.
- The `src` argument refers to an array of reference component type and the `dst` argument refers to an array of primitive component type.

Otherwise, if any of the following is true, an `IndexOutOfBoundsException` is thrown and the destination is not modified:

- The `srcOffset` argument is negative.

- The `dstOffset` argument is negative.
- The `length` argument is negative.
- `srcOffset+length` is greater than `src.length`, the length of the `src` array.
- `dstOffset+length` is greater than `dst.length`, the length of the `dst` array.

Otherwise, if the actual value of any component of the source array from position `srcOffset` through `srcOffset+length-1` cannot be converted to the component type of the destination array by assignment conversion, then an `ArrayStoreException` is thrown. In this case, let k be the smallest nonnegative integer less than `length` such that `src[srcOffset+k]` cannot be converted to the component type of the destination array. When the exception is thrown, the source array components from positions `srcOffset` through `srcOffset+k-1` have been copied to destination array positions `dstOffset` through `dstOffset+k-1` and no other positions of the destination array will have been modified. (Because of the restrictions already itemized, this paragraph effectively applies only to the situation where both arrays have component types that are reference types.)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.19 The Interface java.lang.Runnable

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a new thread. All that is required of such a class is that it implement a method of no arguments called `run`.

```
public interface Runnable {  
    public abstract void run();  
}
```

20.19.1 `public abstract void run()`

The general contract of the method `run` is that it may take any action whatsoever.

If an object implementing interface `Runnable` is used to create a thread ([§20.20](#)), then starting the thread will (normally) lead to the invocation of the object's `run` method in that separately executing thread.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.20 The Class java.lang.Thread

A thread is a single sequential flow of control. Thread objects allow multithreaded Java programming; a single Java Virtual Machine can execute many threads in an interleaved or concurrent manner.

In the method descriptions that follow, it is very important to distinguish among "the current thread" (the thread executing the method), "this `Thread`" (the object for which the method was invoked), and "this thread" (the thread that is represented by the `Thread` object for which the method was invoked).

```
public class Thread implements Runnable {
    public final static int MIN    PRIORITY = 1;
    public final static int MAX    PRIORITY = 10;
    public final static int NORM   PRIORITY = 5;
    public Thread();
    public Thread(String name);
    public Thread(Runnable target);
    public Thread(Runnable target, String name);
    public Thread(ThreadGroup group, String name)
        throws SecurityException, IllegalThreadStateException;
    public Thread(ThreadGroup group, Runnable target)
        throws SecurityException, IllegalThreadStateException;
    public Thread(ThreadGroup group, Runnable target,
        String name)
        throws SecurityException, IllegalThreadStateException;
    public String toString();
    public void checkAccess();
    public void run();
    public void start()
        throws IllegalThreadStateException;
    public final void stop()
        throws SecurityException;
    public final void stop(Throwable o)
        throws SecurityException;
    public final void suspend()
        throws SecurityException;
    public final void resume()
        throws SecurityException;
    public void destroy();
    public final boolean isAlive();
    public void interrupt();
    public static boolean interrupted();
    public boolean isInterrupted();
    public final String getName();
    public final void setName(String name)
        throws SecurityException;
    public final ThreadGroup getThreadGroup();
    public final int getPriority();
    public final void setPriority(int newPriority)
        throws SecurityException, IllegalArgumentException;
    public final boolean isDaemon();
    public final void setDaemon(boolean on)
        throws SecurityException;
    public int countStackFrames();
}
```



```

    public final void join()
        throws InterruptedException;
    public final void join(long millis)
        throws InterruptedException;
    public final void join(long millis, int nanos)
        throws InterruptedException;
    public void interrupt();
    public boolean isInterrupted();
    public static boolean interrupted();
    public static Thread currentThread();
    public static int activeCount();

    // deprecated
    public static int enumerate(Thread tarray[]);

    //
    deprecated
    public static void dumpStack();
    public static void yield();
    public static void sleep(long millis)

        throws InterruptedException;
    public static void sleep(long millis, int nanos)

        throws InterruptedException;
}

```

When a new `Thread` object is created, the thread it represents is not yet active. It is activated when some other thread calls the `start` method ([§20.20.14](#)) of the `Thread` object. This causes the thread represented by the `Thread` object to invoke the `run` method ([§20.20.13](#)) of the `Thread` object. The newly activated thread then remains alive until it stops because one of five things occurs:

- The initial invocation of the `run` method by the newly activated thread completes normally through a normal return from the `run` method.
- The initial invocation of the `run` method by the newly activated thread completes abruptly because an exception was thrown.
- The thread invokes the `stop` method ([§20.20.15](#)) of the `Thread` object (and the security manager ([§20.17.11](#)) approves execution of the `stop` operation).
- Some other thread invokes the `stop` method of the `Thread` object (and the security manager ([§20.17.11](#)) approves execution of the `stop` operation).
- Some thread invokes the `exit` method ([§20.16.2](#)) of class `Runtime` (and the security manager ([§20.17.13](#)) approves execution of the `exit` operation); this stops every thread being run by the Java Virtual Machine that is running the thread that invokes the `exit` method.

As a thread dies, the `notifyAll` method ([§20.1.10](#)) is invoked for the `Thread` object that represents it; this fact is important for the proper operation of the `join` methods ([§20.20.28](#), [§20.20.29](#), [§20.20.30](#)). A thread is also removed from its thread group as it dies. Once a thread has been stopped, it is no longer alive and it cannot be restarted.

A thread that is alive can be *suspended* and *resumed*. A suspended thread is considered to be alive, but it performs no work, makes no progress, executes no virtual machine instructions. Resumption restores a thread to the state of active execution. A thread is suspended when it or another thread calls the `suspend` method ([§20.20.17](#)) of the `Thread` object that represents it (and the security manager ([§20.17.11](#)) approves execution of the `suspend` operation). A thread is resumed when another thread calls the `resume` method ([§20.20.18](#)) of the `Thread` object that represents it (and the

security manager ([§20.17.11](#)) approves execution of the `resume` operation).

Every thread has a *priority*. When there is competition for processing resources, threads with higher priority are generally executed in preference to threads with lower priority. Such preference is not, however, a guarantee that the highest priority thread will always be running, and thread priorities cannot be used to implement mutual exclusion. When code running in some thread creates a new `Thread` object, the newly created thread has its priority initially set equal to the priority of the creating thread. But the priority of a thread *T* may be changed at any time if some thread invokes the `setPriority` method of the `Thread` object that represents *T* (and the security manager ([§20.17.11](#)) approves execution of the `setPriority` operation).

Each thread may or may not be marked as a *daemon*. When code running in some thread creates a new `Thread` object, the newly created thread is a daemon thread if and only if the creating thread is a daemon. But the daemonhood of a thread *T* may be changed before it is activated if some other thread invokes the `setDaemon` method of the `Thread` object that represents *T* (and the security manager ([§20.17.11](#)) approves execution of the `setDaemon` operation).

When a Java Virtual Machine starts up, there is usually a single non-daemon thread, which typically begins by invoking the method `main` of some designated class. The Java Virtual Machine continues to execute threads according to the thread execution model until all threads that are not daemon threads have stopped.

There are two ways to create a new thread of execution. One is to declare some class to be a subclass of `Thread`; this subclass should override the `run` method of class `Thread`. An instance of the subclass can then be created and started. For example, consider code for a thread whose job is to compute primes larger than a stated value:

```
class PrimeThread extends Thread {

    long minPrime;

    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
        ...
    }

}
```

The following code would then create a thread and start it running:

```
PrimeThread p = new PrimeThread(143);
p.start();
```

The other way to create a thread is to declare some class to implement the `Runnable` interface, which also requires that the class implement the `run` method. An instance of the class can then be created, used to create a `Thread`, and started. The same example in this other style looks like this:


```

class PrimeRun implements Runnable {

    long minPrime;

    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
        ...
    }

}

```

The following code would then create a thread and start it running:

```

PrimeRun p = new PrimeRun(143);
new Thread(p).start();

```

Every thread has a name, which is a `String`, for identification purposes. More than one thread may have the same name. If a name is not specified when a thread is created, a new name is generated for it.

Every thread that has not yet been stopped belongs to a thread group ([§20.21](#)). A thread can always create a new thread in its own thread group. To create a thread in some other thread group requires the approval of the `checkAccess` method ([§20.21.4](#)) of that thread group, which forwards the decision to the security manager ([§20.17.11](#)).

20.20.1 `public final static int MIN_PRIORITY = 1;`

The constant value of this field is 1, the smallest allowed priority for a thread.

20.20.2 `public final static int MAX_PRIORITY = 10;`

The constant value of this field is 10, the largest allowed priority value for a thread.

20.20.3 `public final static int NORM_PRIORITY = 5;`

The constant value of this field is 5, the normal priority for a thread that is not a daemon.

20.20.4 `public Thread()`

This constructor initializes a newly created `Thread` object so that it has no separate run object, has a newly generated name, and belongs to the same thread group as the thread that is creating the new thread.

This constructor has exactly the same effect as the explicit constructor call `this(null, null, gname)` ([§20.20.10](#)), where *gname* is a newly generated name. Automatically generated names are

of the form "Thread-" + n , where n is an integer.

20.20.5 `public Thread(String name)`

This constructor initializes a newly created `Thread` object so that it has no separate run object, has the specified `name` as its name, and belongs to the same thread group as the thread that is creating the new thread.

This constructor has exactly the same effect as the explicit constructor call `this(null, null, name)` ([§20.20.10](#)).

20.20.6 `public Thread(Runnable runObject)`

This constructor initializes a newly created `Thread` object so that it has the given `runObject` as its separate run object, has a newly generated name, and belongs to the same thread group as the thread that is creating the new thread.

This constructor has exactly the same effect as the explicit constructor call `this(null, runObject, gname)` ([§20.20.10](#)) where `gname` is a newly generated name. Automatically generated names are of the form "Thread-" + n where n is an integer.

20.20.7 `public Thread(Runnable runObject, String name)`

This constructor initializes a newly created `Thread` object so that it has the given `runObject` as its separate run object, has the specified `name` as its name, and belongs to the same thread group as the thread that is creating the new thread.

This constructor has exactly the same effect as the explicit constructor call `this(null, runObject, name)` ([§20.20.10](#)).

20.20.8 `public Thread(ThreadGroup group, String name)`
throws `SecurityException`, `IllegalThreadStateException`

First, if `group` is not `null`, the `checkAccess` method ([§20.21.4](#)) of that thread group is called with no arguments.

This constructor initializes a newly created `Thread` object so that it has no separate run object, has the specified `name` as its name, and belongs to the thread group referred to by `group` (but if `group` is `null`, then the new thread will belong to the same thread group as the thread that is creating the new thread).

If `group` is a `ThreadGroup` that has been destroyed by method `destroy` ([§20.21.11](#)), then an `IllegalThreadStateException` is thrown.

This constructor has exactly the same effect as the explicit constructor call `Thread(group, null, name)` ([§20.20.10](#)).

20.20.9 `public Thread(ThreadGroup group, Runnable runObject)`
throws `SecurityException`, `IllegalThreadStateException`

First, if `group` is not `null`, the `checkAccess` method ([§20.21.4](#)) of that thread group is called with no arguments.

This constructor initializes a newly created `Thread` object so that it has the given `runObject` as its

thread,.

This method is called by methods `stop` of no arguments ([§20.20.15](#)), `stop` of one argument ([§20.20.16](#)), `suspend` ([§20.20.17](#)), `resume` ([§20.20.18](#)), `setName` ([§20.20.20](#)), `setPriority` ([§20.20.23](#)), and `setDaemon` ([§20.20.25](#)).

20.20.13 `public void run()`

The general contract of this method is that it should perform the intended action of the thread.

The `run` method of class `Thread` simply calls the `run` method of the separate run object, if there is one; otherwise, it does nothing.

20.20.14 `public void start()`
throws `IllegalThreadStateException`

Invoking this method causes this thread to begin execution; this thread calls the `run` method of this `Thread` object. The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the thread represented by this `Thread` object (which executes its `run` method).

20.20.15 `public final void stop()`
throws `SecurityException`

First, the `checkAccess` method ([§20.20.12](#)) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

This thread is forced to complete abnormally whatever it was doing and to throw a `ThreadDeath` object as an exception. For this purpose, this thread is resumed if it had been suspended, and is awakened if it had been asleep.

It is permitted to stop a thread that has not yet been started. If the thread is eventually started, it will immediately terminate.

User code should not normally try to catch `ThreadDeath` unless some extraordinary cleanup operation is necessary (note that the process of throwing a `ThreadDeath` exception *will* cause *finally* clauses of `try` statements to be executed before the thread officially dies). If a `catch` clause does catch a `ThreadDeath` object, it is important to rethrow the object so that the thread will actually die. The top-level error handler that reacts to otherwise uncaught exceptions will not print a message or otherwise signal or notify the user if the uncaught exception is an instance of `ThreadDeath`.

20.20.16 `public final void stop(Throwable thr)`
throws `SecurityException`, `NullPointerException`

First, the `checkAccess` method ([§20.20.12](#)) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

If the argument `thr` is null, then a `NullPointerException` is thrown (in the current thread).

This thread is forced to complete abnormally whatever it was doing and to throw the `Throwable` object `thr` as an exception. For this purpose, this thread is resumed if it had been suspended, and is awakened if it had been asleep. This is an unusual action to take; normally, the `stop` method that

takes no arguments [\(§20.20.15\)](#) should be used.

It is permitted to stop a thread that has not yet been started. If the thread is eventually started, it will immediately terminate.

20.20.17 `public final void suspend()
throws SecurityException`

First, the `checkAccess` method [\(§20.20.12\)](#) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

If this thread is alive [\(§20.20.26\)](#), it is suspended and makes no further progress unless and until it is resumed. It is permitted to suspend a thread that is already in a suspended state; it remains suspended. Suspensions are not tallied; even if a thread is suspended more than once, only one call to `resume` is required to resume it.

20.20.18 `public final void resume()
throws SecurityException`

First, the `checkAccess` method [\(§20.20.12\)](#) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

If this thread is alive [\(§20.20.26\)](#) but suspended, it is resumed and is permitted to make progress in its execution. It is permitted to resume a thread that has never been suspended or has already been resumed; it continues to make progress in its execution. Resumptions are not tallied; even if a thread is resumed more than once, only one call to `suspend` is required to suspend it.

20.20.19 `public final String getName()`

The current name of this `Thread` object is returned as a `String`.

20.20.20 `public final void setName(String name)
throws SecurityException`

First, the `checkAccess` method [\(§20.20.12\)](#) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

The name of this `Thread` object is changed to be equal to the argument `name`.

20.20.21 `public final ThreadGroup getThreadGroup()`

If this thread is alive, this method returns a reference to the `ThreadGroup` object that represents the thread group to which this thread belongs. If this thread has died (has been stopped), this method returns `null`.

20.20.22 `public final int getPriority()`

The current priority of this `Thread` object is returned.

20.20.23 `public final void setPriority(int newPriority)
throws SecurityException, IllegalArgumentException`

First, the `checkAccess` method ([§20.20.12](#)) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

If the `newPriority` argument is less than `MIN_PRIORITY` ([§20.20.1](#)) or greater than `MAX_PRIORITY` ([§20.20.2](#)), then an `IllegalArgumentException` is thrown.

Otherwise, the priority of this `Thread` object is set to the smaller of the specified `newPriority` and the maximum permitted priority ([§20.21.12](#)) of the thread's thread group ([§20.20.21](#)).

20.20.24 `public final boolean isDaemon()`

The result is `true` if and only if this thread is marked as a daemon thread.

20.20.25 `public final void setDaemon(boolean on)`
throws `SecurityException`, `IllegalThreadStateException`

First, the `checkAccess` method ([§20.20.12](#)) of this `Thread` object is called with no arguments. This may result in throwing a `SecurityException` (in the current thread).

If this thread is alive, an `IllegalThreadStateException` is thrown. Otherwise, this thread is marked as being a daemon thread if the argument is `true`, and as not being a daemon thread if the argument is `false`.

20.20.26 `public final boolean isAlive()`

The result is `true` if and only if this thread is alive (it has been started and has not yet died).

20.20.27 `public int countStackFrames()`

This method returns the number of Java Virtual Machine stack frames currently active for this thread.

20.20.28 `public final void join()` throws `InterruptedException`

This method causes the current thread to wait (using the `wait` method ([§20.1.6](#)) of class `Object`) until this thread is no longer alive.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then the wait is ended and an `InterruptedException` is thrown.

20.20.29 `public final void join(long millis)`
throws `InterruptedException`

This method causes the current thread to wait (using the `wait` method ([§20.1.7](#)) of class `Object`) until either this thread is no longer alive or a certain amount of real time has elapsed, more or less.

The amount of real time, measured in milliseconds, is given by `millis`. If `millis` is zero, however, then real time is not taken into consideration and this method simply waits until this thread is no longer alive.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then the wait is ended and an `InterruptedException` is thrown.

20.20.30 `public final void join(long millis, int nanos)`
 throws `InterruptedException`

This method causes the current thread to wait (using the `wait` method ([§20.1.8](#)) of class `Object`) until either this thread is no longer alive or a certain amount of real time has elapsed, more or less.

The amount of real time, measured in nanoseconds, is given by:

`1000000*millis+nanos`

In all other respects, this method does the same thing as the method `join` of one argument ([§20.20.29](#)). In particular, `join(0, 0)` means the same thing as `join(0)`.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then the wait is ended and an `InterruptedException` is thrown.

20.20.31 `public void interrupt()`

An interrupt request is posted for this thread. This thread does not necessarily react immediately to the interrupt, however. If this thread is waiting, it is awakened and it then throws an `InterruptedException`.

[This method is scheduled for introduction in Java version 1.1.]

20.20.32 `public boolean isInterrupted()`

The result is `true` if and only if an interrupt request has been posted for this thread.

[This method is scheduled for introduction in Java version 1.1.]

20.20.33 `public static boolean interrupted()`

The result is `true` if and only if an interrupt request has been posted for the current thread.

[This method is scheduled for introduction in Java version 1.1.]

20.20.34 `public static Thread currentThread()`

The `Thread` object that represents the current thread is returned.

20.20.35 `public static int activeCount()`

This method returns the number of active threads in the thread group to which the current thread belongs. This count includes threads in subgroups of that thread group. This is the same as the value of the expression:

`Threads.currentThread().getThreadGroup().activeCount()`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Instead, an expression equivalent to:


```
Threads.currentThread().getThreadGroup().allThreadsCount()
```

should be used. See the method `allThreadsCount` of class `ThreadGroup`.]

20.20.36 `public static int enumerate(Thread tarray[])`

The active threads in the thread group to which the current thread belongs, including threads in subgroups of that thread group, are enumerated and their `Thread` objects are put into the array `tarray`. The number of threads actually put into the array is returned. Call this value n ; then the threads have been put into elements 0 through $n-1$ of `tarray`. If the number of threads exceeds the length of `tarray`, then some of the threads, `tarray.length` of them, are chosen arbitrarily and used to fill the array `tarray`.

[This method is deprecated for use in new code after Java version 1.1 becomes available. Instead, an expression equivalent to:

```
Threads.currentThread().getThreadGroup().allThreads()
```

should be used. See the method `allThreads` of class `ThreadGroup`.]

20.20.37 `public static void dumpStack()`

This is a utility method that makes it easy to print a stack dump for the current thread. It is equivalent in effect to:

```
new Exception("Stack trace").printStackTrace()
```

See the `printStackTrace` method ([§20.22.6](#)) of class `Throwable`.

20.20.38 `public static void yield()`

This method causes the current thread to yield, allowing the thread scheduler to choose another runnable thread for execution.

20.20.39 `public static void sleep(long millis)`
throws `InterruptedException`

This method causes the current thread to yield and not to be scheduled for further execution until a certain amount of real time has elapsed, more or less.

The amount of real time, measured in milliseconds, is given by `millis`.

If the current thread is interrupted ([§20.20.31](#)) by another thread while it is waiting, then the sleep is ended and an `InterruptedException` is thrown.

20.20.40 `public static void sleep(long millis, int nanos)`
throws `InterruptedException`

This method causes the current thread to yield and not to be scheduled for further execution until a certain amount of real time has elapsed, more or less.

The amount of real time, measured in nanoseconds, is given by:

```
1000000*millis+nanos
```

In all other respects, this method does the same thing as the method `sleep` of one argument [\(§20.20.39\)](#). In particular, `sleep(0, 0)` means the same thing as `sleep(0)`.

If the current thread is interrupted [\(§20.20.31\)](#) by another thread while it is waiting, then the sleep is ended and an `InterruptedException` is thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


20.21 The Class java.lang.ThreadGroup

A thread group is a set of threads and thread groups. Every thread belongs to exactly one thread group, and every thread group but one (called the "system thread group") belongs to some other thread group. Thus thread groups form a tree with the system thread group at the root.

Thread groups provide a way to manage threads and to impose security boundaries; for example, a thread may always create a new thread within its own thread group, but creating a thread in another thread group requires the approval of the security manager ([§20.17](#)), as does the creation of a new thread group.

```
public class ThreadGroup {
    public ThreadGroup(String name)
        throws SecurityException;
    public ThreadGroup(ThreadGroup parent, String name)
        throws NullPointerException, SecurityException,
        IllegalArgumentException;
    public String toString();
    public final void checkAccess();
    public final String getName();
    public final ThreadGroup getParent();
    public final boolean parentOf(ThreadGroup g);
    public final void stop()
        throws SecurityException;
    public final void suspend()
        throws SecurityException;
    public final void resume()
        throws SecurityException;
    public final void destroy()
        throws SecurityException, IllegalArgumentException;
    public final int getMaxPriority();
    public final void setMaxPriority(int newMaxPriority)
        throws SecurityException, IllegalArgumentException;
    public final boolean isDaemon();
    public final void setDaemon(boolean daemon)
        throws SecurityException;
    public int threadsCount();
    public int allThreadsCount() ;
    public int groupsCount();
    public int allGroupsCount();
    public Thread[] threads();
    public ThreadGroup[] groups();
    public Thread[] allThreads();
    public ThreadGroup[] allGroups();
    public int activeCount();

    // deprecated
    public int activeGroupCount();

    // deprecated
    public int enumerate(Thread list[]);

    // deprecated
    public int enumerate(Thread list[],

    // deprecated
        boolean recurse);
    public int enumerate(ThreadGroup list[]);

    // deprecated
```



```

    public int enumerate(ThreadGroup list[],
                           // deprecated
                           boolean recurse);
    public void list();
    public void uncaughtException(Thread t, Throwable e);
}

```

Every thread group has a *maximum priority*. The priority of a thread cannot be set [\(\\$20.20.23\)](#) higher than the maximum priority of its thread group.

Each thread group may or may not be marked as a *daemon*. When a new `ThreadGroup` object is created, the newly created thread group is marked as a daemon thread group if and only if the thread group to which it belongs is currently a daemon thread group. But the daemonhood of a thread group `G` may be changed at any time by calling the `setDaemon` method of the `ThreadGroup` object that represents `G` (provided that the security manager [\(\\$20.17.12\)](#) approves execution of the `setDaemon` operation).

Every thread group has a name, which is a `String`, for identification purposes. More than one thread group may have the same name.

Creation of a thread group requires the approval of the `checkAccess` method [\(\\$20.21.4\)](#) of its proposed parent thread group, which forwards the decision to the security manager [\(\\$20.17.11\)](#).

20.21.1 `public ThreadGroup(String name)`
throws `SecurityException`

First, the `checkAccess` method [\(\\$20.21.4\)](#) of the thread group to which the current thread belongs is called with no arguments.

This constructor initializes a newly created `ThreadGroup` object so that it has the specified `name` as its name and belongs to the same thread group as the thread that is creating the new thread group.

This constructor has exactly the same effect as the explicit constructor call
`this(Thread.currentThread().getThreadGroup(), name)` [\(\\$20.21.2\)](#).

20.21.2 `public ThreadGroup(ThreadGroup parent, String name)`
throws `NullPointerException`, `SecurityException`,
`IllegalThreadStateException`

First, the `checkAccess` method [\(\\$20.21.4\)](#) of the `parent` thread group is called with no arguments.

If `parent` is `null`, then a `NullPointerException` is thrown. If `parent` is a `ThreadGroup` that has been destroyed by method `destroy` [\(\\$20.21.11\)](#), then an `IllegalThreadStateException` is thrown.

This constructor initializes a newly created `ThreadGroup` object so that it has the specified `name` as its name and belongs to the thread group represented by `parent`.

The maximum priority for the newly created thread group is set equal to the maximum priority of `parent`. The method `setMaxPriority` [\(\\$20.21.13\)](#) may be used to change the maximum priority to a lower value.

The newly created thread group is initially marked as being a daemon thread group if and only if `parent` is a daemon thread group. The method `setDaemon` [\(\\$20.21.15\)](#) may be used to change whether or not a thread group is a daemon thread group.

20.21.3 `public String toString()`

The returned value is a concatenation of the following six strings:

- The name of the class of this thread group object
- "[name="
- The name ([§20.21.5](#)) of this thread group
- ",maxpri="
- The current maximum priority ([§20.21.12](#)) for this thread group, as a decimal numeral
- "]"

All literal characters mentioned above are from the ASCII subset of Unicode.

Overrides the `toString` method of `Object` ([§20.1.3](#)).

20.21.4 `public final void checkAccess()`

If there is a security manager, its `checkAccess` method ([§20.17.12](#)) is called with this `ThreadGroup` object as its argument. This may result in throwing, in the current thread, a `SecurityException`.

This method is called by methods `stop` ([§20.21.8](#)), `suspend` ([§20.21.9](#)), `resume` ([§20.21.10](#)), `destroy` ([§20.21.11](#)), `setMaxPriority` ([§20.21.13](#)), and `setDaemon` ([§20.21.15](#)).

20.21.5 `public final String getName()`

The current name of this `ThreadGroup` object is returned as a `String`.

20.21.6 `public final ThreadGroup getParent()`

This method returns the `ThreadGroup` object that represents the thread group to which this thread group belongs. If this thread group is the system thread group, which is at the root of the thread group hierarchy, then `null` is returned.

20.21.7 `public final boolean parentOf(ThreadGroup g)`

This method returns `true` if and only if either this thread group is `g` or this method is `true` when applied to the parent of `g`. In other words, this method says whether this thread group is an ancestor of `g` or perhaps `g` itself.

(This method arguably is misnamed; a more accurate, if clumsy and abstruse, name would be `parentOfReflexiveTransitiveClosure`.)

20.21.8 `public final void stop() throws SecurityException`

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

Every thread in this thread group or any of its subgroups is stopped. More precisely, the method `stop` is called for every `ThreadGroup` and every `Thread` ([§20.20.15](#)) that belongs to this `ThreadGroup`.

20.21.9 `public final void suspend() throws SecurityException`

*Suspended under a twilight canopy,
We'll search the clouds for a star to guide us.*
--Jim Webb, Up, Up and Away (1967)

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

Every thread in this thread group or any of its subgroups is suspended. More precisely, the method `suspend` is called for every `ThreadGroup` and every `Thread` ([§20.20.17](#)) that belongs to this `ThreadGroup`.

20.21.10 `public final void resume() throws SecurityException`

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

Every thread in this thread group or any of its subgroups is resumed. More precisely, the method `resume` is called for every `ThreadGroup` and every `Thread` ([§20.20.18](#)) that belongs to this `ThreadGroup`.

20.21.11 `public final void destroy()
throws SecurityException, IllegalThreadStateException`

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

This thread group is destroyed. If it has already been destroyed, or if any threads belong to it directly, then an `IllegalThreadStateException` is thrown. Otherwise, this method is called recursively for every thread group that belongs to this thread group, and this thread group is removed from its parent thread group.

A thread group that is currently marked as a daemon thread group is destroyed automatically if both of the following conditions are true:

- A thread or thread group has just been removed from it (because the thread has died or the thread group has been destroyed).
- The thread group now contains no more threads or thread groups.

20.21.12 `public final int getMaxPriority()`

The current maximum priority of this `ThreadGroup` object is returned.

20.21.13 `public final void setMaxPriority(int newMaxPriority)
throws SecurityException, IllegalArgumentException`

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

If the `newMaxPriority` argument is less than `MIN_PRIORITY` ([§20.20.1](#)) or greater than

`MAX_PRIORITY` ([§20.20.2](#)), then an `IllegalArgumentException` is thrown.

Otherwise, the priority of this `ThreadGroup` object is set to the smaller of the specified `newMaxPriority` and the maximum permitted priority ([§20.21.12](#)) of the parent of this thread group ([§20.21.12](#)). (If this thread group is the system thread group, which has no parent, then its maximum priority is simply set to `newMaxPriority`.) Then this method is called recursively, with `newMaxPriority` as its argument, for every thread group that belongs to this thread group.

20.21.14 `public final boolean isDaemon()`

The result is `true` if and only if this thread group is currently marked as a daemon thread group.

20.21.15 `public final void setDaemon(boolean daemon)`
throws `SecurityException`

First, the `checkAccess` method ([§20.21.4](#)) of this `ThreadGroup` object is called with no arguments. This may result in a `SecurityException` being thrown (in the current thread).

This thread group is marked as being a daemon thread group if the argument is `true`, and as not being a daemon thread group if the argument is `false`.

20.21.16 `public int threadsCount()`

This method returns the number of threads that directly belong to this thread group.

20.21.17 `public int allThreadsCount()`

This method returns the number of threads that belong to this thread group or to any of its subgroups.

20.21.18 `public int groupsCount()`

This method returns the number of thread groups that directly belong to this thread group.

20.21.19 `public int allGroupsCount()`

This method returns the number of thread groups that belong to this thread group or to any of its subgroups.

20.21.20 `public Thread[] threads()`

This method returns a newly created array containing the `Thread` objects for all threads that directly belong to this thread group.

20.21.21 `public Thread[] allThreads()`

This method returns a newly created array containing the `Thread` objects for all threads that belong to this thread group or to any of its subgroups.

20.21.22 `public ThreadGroup[] groups()`

This method returns a newly created array containing the `ThreadGroup` objects for all thread groups that directly belong to this thread group.

20.21.23 `public ThreadGroup[] allGroups()`

This method returns a newly created array containing the `ThreadGroup` objects for all thread groups that belong to this thread group or to any of its subgroups.

20.21.24 `public int activeCount()`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the equivalent method `allThreadsCount` instead.]

20.21.25 `public int activeGroupCount()`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the equivalent method `allGroupsCount` instead.]

20.21.26 `public int enumerate(Thread list[])`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the method `allThreads` instead.]

20.21.27 `public int enumerate(Thread list[], boolean recurse)`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the method `threads` or `allThreads` instead.]

20.21.28 `public int enumerate(ThreadGroup list[])`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the method `allGroups` instead.]

20.21.29 `public int enumerate(ThreadGroup list[], boolean recurse)`

[This method is deprecated for use in new code after Java version 1.1 becomes available. Use the method `groups` or `allGroups` instead.]

20.21.30 `public void list()`

This method prints a detailed description of this thread group to the output stream `System.out` ([§20.18.2](#)). It is intended as a convenient utility for debugging.

The output is a series of lines; each line contains some space characters (for indentation) followed by the `toString` representation of one thread ([§20.20.11](#)) or one thread group ([§20.21.3](#)).

The first line gives the `toString` representation for this thread group, with no indentation spaces. Following lines are then generated by a recursive rule: whenever a line is printed for a thread group *G* with *n* leading spaces, it is immediately followed by one line for each thread that directly belongs to *G*, with {ewc msdncd, EWGraphic, LNG21x 0 /a "langref.BMP"} spaces of indentation;

then one line is printed for each thread group that directly belongs to *G*, with {ewc msdncd, EWGraphic, LNG21x 1 /a "langref.BMP"} spaces of indentation, using the recursive case.

20.21.31 `public void uncaughtException(Thread t, Throwable e)`

The general contract of `uncaughtException` is that it is called whenever a thread that belongs directly to this thread group dies because an exception was thrown in that thread and not caught. The arguments are the `Thread` object for the thread in question and the `Throwable` object that was thrown. The `uncaughtException` method may then take any appropriate action.

The call to `uncaughtException` is performed by the thread that failed to catch the exception, so *t* is the current thread. The call to `uncaughtException` is the last action of the thread before it dies. If the call to `uncaughtException` itself results in an (uncaught) exception, this fact is ignored and the thread merely goes on to die.

The method `uncaughtException` defined by class `ThreadGroup` takes one of two actions. If this thread group has a parent thread group, then this method is invoked for that parent thread group, with the same arguments. If this thread group is the system thread group (which has no parent), then if the exception *e* is not an instance of `ThreadDeath` (§20.22), a stack trace (§20.22.6) for *e* is printed on the error output stream that is the value of the field `System.err` (§20.18.3).

Subclasses of `ThreadGroup` may override the `uncaughtException` method.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.22 Classes for Exceptions and Errors

The `throw` statement ([§14.16](#)) is permitted to throw only instances of the class `Throwable` and its subclasses. Instances of two subclasses, `Error` and `Exception`, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

The following list shows the hierarchical relationships of all the exception classes predefined in package `java.lang` by the Java language:

```
Throwable
  Error
    LinkageError
      ClassCircularityError
      ClassFormatError
      ExceptionInInitializerError

      IncompatibleClassChangeError
        AbstractMethodError
        IllegalAccessError
        InstantiationError
        NoSuchFieldError
        NoSuchMethodError
      NoClassDefFoundError
      UnsatisfiedLinkError
      VerifyError
    VirtualMachineError
      InternalError
      OutOfMemoryError
      StackOverflowError
      UnknownError
  ThreadDeath
  Exception
    ClassNotFoundException
    CloneNotSupportedException
    IllegalAccessException
    InstantiationException
    InterruptedException
    RuntimeException
      ArithmeticException
      ArrayStoreException
      ClassCastException
      IllegalArgumentException
        IllegalThreadStateException
      NumberFormatException
      IllegalMonitorStateException
      IndexOutOfBoundsException
      NegativeArraySizeException
      NullPointerException
      SecurityException
```

By convention, class `Throwable` and all its subclasses have two constructors, one that takes no

arguments and one that takes a `String` argument that can be used to produce an error message. This is true of all the classes shown above, with one exception: `ExceptionInInitializerError`. These predefined classes otherwise have no new content; they merely inherit methods from class `Throwable`.

```
public class Throwable {
    public Throwable();
    public Throwable(String message);
    public String toString();
    public String getMessage();
    public Throwable fillInStackTrace();
    public void printStackTrace();
    public void printStackTrace(java.io.PrintStream s);
}
```

20.22.1 `public Throwable()`

This constructor initializes a newly created `Throwable` object with `null` as its error message string. Also, the method `fillInStackTrace` ([§20.22.5](#)) is called for this object.

20.22.2 `public Throwable(String message)`

This constructor initializes a newly created `Throwable` object by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)). Also, the method `fillInStackTrace` ([§20.22.5](#)) is called for this object.

20.22.3 `public String getMessage()`

If this `Throwable` object was created with an error message string ([§20.22.2](#)), then a reference to that string is returned.

If this `Throwable` object was created with no error message string ([§20.22.1](#)), then `null` is returned.

20.22.4 `public String toString()`

If this `Throwable` object was created with an error message string ([§20.22.2](#)), then the result is the concatenation of three strings:

- The name of the actual class of this object
- `": "` (a colon and a space)
- The result of the `getMessage` method ([§20.22.3](#)) for this object

If this `Throwable` object was created with no error message string ([§20.22.1](#)), then the name of the actual class of this object is returned.

20.22.5 `public Throwable fillInStackTrace()`

This method records within this `Throwable` object information about the current state of the stack

frames for the current thread.

20.22.6 `public void printStackTrace()`

This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err` ([§20.18.3](#)). The first line of output contains the result of the `toString` method ([§20.22.4](#)) for this object. Remaining lines represent data previously recorded by the method `fillInStackTrace` ([§20.22.5](#)). The format of this information depends on the implementation, but the following example may be regarded as typical:

```
java.lang.NullPointerException
  at MyClass.mash(MyClass.java:9)
  at MyClass.crunch(MyClass.java:6)
  at MyClass.main(MyClass.java:3)
```

This example was produced by running the program:

```
class MyClass {

    public static void main(String[] argv) {
        crunch(null);
    }

    static void crunch(int[] a) {
        mash(a);
    }

    static void mash(int[] b) {
        System.out.println(b[0]);
    }

}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

20.23 The Class java.lang.ExceptionInInitializerError

An `ExceptionInInitializerError` is thrown to indicate that an exception occurred during evaluation of a static initializer or the initializer for a static variable ([§12.4.2](#)).

```
public class ExceptionInInitializerError
    extends RuntimeException {
    public ExceptionInInitializerError();
    public ExceptionInInitializerError(String s);
    public ExceptionInInitializerError(Throwable thrown);
    public Throwable getException();
}
```

20.23.1 `public ExceptionInInitializerError()`

This constructor initializes a newly created `ExceptionInInitializerError` with `null` as its error message string and with a no saved throwable object.

20.23.2 `public ExceptionInInitializerError(String s)`

This constructor initializes a newly created `ExceptionInInitializerError` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)). There is no saved throwable object.

20.23.3 `public ExceptionInInitializerError(Throwable thrown)`

This constructor initializes a newly created `ExceptionInInitializerError` by saving a reference to the `Throwable` object `thrown` for later retrieval by the `getException` method ([§20.22.3](#)). The error message string is set to `null`.

20.23.4 `public Throwable getException(Throwable thrown)`

The saved throwable object of this `ExceptionInInitializerError` is returned; `null` is returned if this `ExceptionInInitializerError` has no saved throwable object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

The Package java.util

The `java.util` package contains various utility classes and interfaces.

Notable among these utilities is the `Enumeration` interface. An object that implements this interface will generate a series of items, delivering them on demand, one by one. Container classes such as `Dictionary` and `Vector` provide one or more methods that return an `Enumeration`.

A `BitSet` contains an indexed collection of bits that may be used to represent a set of nonnegative integers.

The class `Date` provides a convenient way to represent and manipulate time and date information. Dates may be constructed from a year, month, day of month, hour, minute, and second, and those six components, as well as the day of the week, may be extracted from a date. Time zones and daylight saving time are properly accounted for.

The abstract class `Dictionary` represents a collection of key-value pairs and allows a value to be fetched given the key. The class `Hashtable` is one concrete implementation of `Dictionary`. The class `Properties` extends `Hashtable` by allowing one table to provide default values for another and by providing standard means for reading entries from files and writing entries to files.

The class `Observable` provides a mechanism for notifying other objects, called "observers," whenever an `Observable` object is changed. An observer object may be any object that implements the `Observer` interface. (This notification mechanism is distinct from that provided by the `wait` and `notify` methods of class `Object` (§20.1) and is not connected with the thread scheduling mechanism.)

The class `Random` provides an extensive set of methods for pseudorandomly generating numeric values of various primitive types and with various distributions. Each instance of class `Random` is an independent pseudorandom generator.

A `StringTokenizer` provides an easy way to divide strings into tokens. The set of characters that delimit tokens is programmable. The tokenizing method is much simpler than the one used by the class `java.io.StreamTokenizer`. For example, a `StringTokenizer` does not distinguish among identifiers, numbers, and quoted strings; moreover, it does not recognize and skip comments.

The classes `Vector` and `Stack` are simple container classes that provide extensions to the capabilities of Java arrays. A `Vector`, unlike a Java array, can change its size, and many convenient methods are provided for adding, removing, and searching for items. A `Stack` is a `Vector` with additional operations such as `push` and `pop`.

The hierarchy of classes defined in package `java.util` is as follows. (Classes whose names are shown here in **boldface** are in package `java.util`; the others are in package `java.lang` and are shown here to clarify subclass relationships.)

```
Object      §20.1
  interface Enumeration  §21.1
  BitSet      §21.2
  Date        §21.3
  Dictionary  §21.4
    Hashtable  §21.5
      Properties §21.6
  Observable §21.7
  interface Observer    §21.8
```


Random	<u>§21.9</u>
StringTokenizer	<u>§21.10</u>
Vector	<u>§21.11</u>
Stack	<u>§21.12</u>
Throwable	<u>§20.22</u>
Exception	
RuntimeException	
EmptyStackException	<u>§21.13</u>
NoSuchElementException	<u>§21.14</u>

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.1 The Interface java.util.Enumeration

An object that implements the `Enumeration` interface will generate a series of elements, one at a time. Successive calls to the `nextElement` method will return successive elements of the series.

```
public interface Enumeration {
    public boolean hasMoreElements();
    public Object nextElement() throws NoSuchElementException;
}
```

21.1.1 `public boolean hasMoreElements()`

The result is `true` if and only if this enumeration object has at least one more element to provide.

21.1.2 `public Object nextElement()` `throws NoSuchElementException`

If this enumeration object has at least one more element to provide, such an element is returned; otherwise, a `NoSuchElementException` is thrown.

As an example, the following code prints every key in the hashtable `ht` and its length. The method `keys` returns an enumeration that will deliver all the keys, and we suppose that the keys are, in this case, known to be strings:

```
Enumeration e = ht.keys();
while (e.hasMoreElements()) {
    String key = (String)e.nextElement();
    System.out.println(key + " " + key.length());
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.2 The Class java.util.BitSet

*'T is an old maxim in the schools,
That flattery's the food of fools;
Yet now and then your men of wit
Will condescend to take a bit.*
--Jonathan Swift, *Cadenus and Vanessa*

A `BitSet` object is a set of bits that grows as needed. The bits of a `BitSet` are indexed by nonnegative integers. Each bit can be individually examined, set, or cleared. One `BitSet` may be used to modify the contents of another `BitSet` through logical AND, logical inclusive OR, and logical exclusive OR operations.

```
public final class BitSet implements Cloneable {  
    public BitSet();  
    public BitSet(int nbits);  
    public String toString();  
    public boolean equals(Object obj)  
    public int hashCode();  
    public Object clone();  
    public boolean get(int bitIndex);  
    public void set(int bitIndex);  
    public void clear(int bitIndex);  
    public void and(BitSet set);  
    public void or(BitSet set);  
    public void xor(BitSet set);  
    public int size();  
}
```

21.2.1 `public BitSet()`

This constructor initializes a newly created `BitSet` so that all bits are clear.

21.2.2 `public BitSet(int nbits)`

This constructor initializes a newly created `BitSet` so that all bits are clear. Enough space is reserved to explicitly represent bits with indices in the range 0 through `nbits-1`.

21.2.3 `public String toString()`

For every index for which this `BitSet` contains a bit in the set state, the decimal representation of that index is included in the result. Such indices are listed in order from lowest to highest, separated by ", " (a comma and a space) and surrounded by braces, resulting in the usual mathematical notation for a set of integers.

Overrides the `toString` method of `Object` (§20.1.2).

Example:


```
BitSet drPepper = new BitSet();
```

Now `drPepper.toString()` returns "{}".

```
drPepper.set(2);
```

Now `drPepper.toString()` returns "{2}".

```
drPepper.set(4);  
drPepper.set(10);
```

Now `drPepper.toString()` returns "{2, 4, 10}".

21.2.4 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `BitSet` object such that, for every nonnegative `int` index `k`:

```
((BitSet) obj).get(k) == this.get(k)
```

Overrides the `equals` method of `Object` ([§20.1.3](#)).

21.2.5 `public int hashCode()`

The hash code depends only on which bits have been set within this `BitSet`. The algorithm used to compute it may be described as follows.

Suppose the bits in the `BitSet` were to be stored in an array of `long` integers called, say, `bits`, in such a manner that bit `k` is set in the `BitSet` (for nonnegative values of `k`) if and only if the expression:

```
((k >> 6) < bits.length) &&  
((bits[k >> 6] & (1L << (bit & 0x3F))) != 0)
```

is true. Then the following definition of the `hashCode` method would be a correct implementation of the actual algorithm:

```
public synchronized int hashCode() {  
    long h = 1234;  
    for (int i = bits.length; --i >= 0; ) {  
        h ^= bits[i] * (i + 1);  
    }  
    return (int) ((h >> 32) ^ h);  
}
```



```
}
```

Note that the hash code value changes if the set of bits is altered.

Overrides the `hashCode` method of `Object` (§20.1.4).

21.2.6 `public Object clone()`

Cloning this `BitSet` produces a new `BitSet` that is equal to it.

Overrides the `clone` method of `Object` (§20.1.5).

21.2.7 `public boolean get(int bit)`

The result is `true` if the bit with index `bit` is currently set in this `BitSet`; otherwise, the result is `false`.

If `bit` is negative, an `IndexOutOfBoundsException` is thrown.

21.2.8 `public void set(int bit)`

The bit with index `bit` in this `BitSet` is changed to the "set" (`true`) state.

If `bit` is negative, an `IndexOutOfBoundsException` is thrown.

If `bit` is not smaller than the value that would be returned by the `size` method (§21.2.13), then the size of this `BitSet` is increased to be larger than `bit`.

21.2.9 `public void clear(int bit)`

The bit with index `bit` in this `BitSet` is changed to the "clear" (`false`) state.

If `bit` is negative, an `IndexOutOfBoundsException` is thrown.

If `bit` is not smaller than the value that would be returned by the `size` method (§21.2.13), then the size of this `BitSet` is increased to be larger than `bit`.

21.2.10 `public void and(BitSet set)`

This `BitSet` may be modified by clearing some of its bits. For every nonnegative `int` index `k`, bit `k` of this `BitSet` is cleared if bit `k` of `set` is clear.

21.2.11 `public void or(BitSet set)`

This `BitSet` may be modified by setting some of its bits. For every nonnegative `int` index `k`, bit `k` of this `BitSet` is set if bit `k` of `set` is set.

21.2.12 `public void xor(BitSet set)`

This `BitSet` may be modified by inverting some of its bits. For every nonnegative `int` index `k`, bit `k` of this `BitSet` is inverted if bit `k` of `set` is set.

21.2.13 `public int size()`

This method returns the number of bits of space actually in use by this `BitSet` to represent bit values.

*At Mooneen he had leaped a place
So perilous that half the astonished meet
Had shut their eyes, and where was it
He rode a race without a bit?*

--William Butler Yeats, *In Memory of Major Robert Gregory* (1919)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.3 The Class java.util.Date

The class `Date` provides a system-independent abstraction of dates and times, to a millisecond precision. Dates may be constructed from a year, month, date (day of month), hour, minute, and second; those six components and the day of the week, may be extracted; and dates may be compared and converted to a readable string.

```
public class Date {
    public Date();
    public Date(long time);
    public Date(int year, int month, int date);
    public Date(int year, int month, int date,
                int hours, int minutes);
    public Date(int year, int month, int date,
                int hours, int minutes, int seconds);
    public Date(String s) throws IllegalArgumentException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public int getYear();
    public void setYear(int year);
    public int getMonth();
    public void setMonth(int month);
    public int getDate();
    public void setDate(int date);
    public int getDay();
    public int getHours();
    public void setHours(int hours);
    public int getMinutes();
    public void setMinutes(int minutes);
    public int getSeconds();
    public void setSeconds(int seconds);
    public long getTime();
    public void setTime(long time);
    public boolean before(Date when);
    public boolean after(Date when);
    public String toLocaleString();
    public String toGMTString();
    public int getTimezoneOffset();
    public static long UTC(int year, int month, int date,
                          int hours, int minutes, int seconds);
    public static long parse(String s)
        throws IllegalArgumentException;
}
```

Examples:

- To print today's date: `System.out.println("today = " + new Date());`
- To find out the day of the week for some particular date, for example, January 16, 1963: `new Date(63, 0, 16).getDay()`

While the `Date` class is intended to reflect UTC (Coordinated Universal Time), it may not do so exactly, depending on the host environment of the Java system. Nearly all modern operating systems

assume that 1 day = {ewc msdncl, EWGraphic, LNG3y 0 /a "langref.BMP"} = 86400 seconds in all cases. In UTC, however, about once every year or two there is an extra second, called a "leap second." The leap second is always added as the last second of the day, and nearly always on December 31 or June 30. For example, the last minute of the year 1995 was 61 seconds long, thanks to an added leap second.

Most computer clocks are currently not accurate enough to be able to reflect the leap-second distinction. Some computer standards are defined in terms of GMT (Greenwich Mean Time), which is equivalent to UT (Universal Time). GMT is the "civil" name for the standard; UT is the "scientific" name for the same standard. The distinction between UTC and UT is that UTC is based on an atomic clock and UT is based on astronomical observations, which for all practical purposes is an invisibly fine hair to split. Because the earth's rotation is not uniform-it slows down and speeds up in complicated ways-UT does not always flow uniformly. Leap seconds are introduced as needed into UTC so as to keep UTC within 0.9 seconds of UT1, which is a version of UT with certain corrections applied. There are other time and date systems as well; for example, the time scale used by GPS (the satellite-based Global Positioning System) is synchronized to UTC but is *not* adjusted for leap seconds. An interesting source of further information is the U. S. Naval Observatory, particularly the Directorate of Time at:

<http://tycho.usno.navy.mil>

and their definitions of "Systems of Time" at:

<http://tycho.usno.navy.mil/systime.html>

In all methods of class `Date` that accept or return year, month, day of month, hours, minutes, and seconds values, the following representations are used:

- A year *y* is represented by the integer {ewc msdncl, EWGraphic, LNG3y 1 /a "langref.BMP"}.
- A month is represented by an integer from 0 to 11; 0 is January, 1 is February, and so on; thus 11 is December.
- A date (day of month) is represented by an integer from 1 to 31 in the usual manner.
- An hour is represented by an integer from 0 to 23. Thus the hour from midnight to 1 AM is hour 0, and the hour from noon to 1 PM is hour 12.
- A minute is represented by an integer from 0 to 59 in the usual manner.
- A second is represented by an integer from 0 to 61. The values 60 and 61 will occur only for leap seconds, and even then only in Java implementations that actually track leap seconds correctly. Because of the manner in which leap seconds are currently introduced, it is extremely unlikely that two leap seconds will occur in the same minute, but this specification follows the date and time conventions for ISO C.

In all cases, arguments given to methods for these purposes need not fall within the indicated ranges; for example, a date may be specified as January 32 and will be interpreted as meaning February 1.

21.3.1 `public Date()`

This constructor initializes a newly created `Date` object so that it represents the instant of time that it was created, measured to the nearest millisecond.

21.3.2 `public Date(long time)`

This constructor initializes a newly created `Date` object so that it represents the instant of time that is `time` milliseconds after the standard base time known as "the epoch," namely 00:00:00 GMT on January 1, 1970. See also the method `currentTimeMillis` ([§20.18.6](#)) of class `System`.

21.3.3 `public Date(int year, int month, int date)`

This constructor initializes a newly created `Date` object so that it represents midnight at the beginning of the day specified by the `year`, `month`, and `date` arguments, in the local time zone. Thus, it has the same effect as the constructor call ([§21.3.5](#)):

```
Date(year, month, date, 0, 0, 0)
```

21.3.4 `public Date(int year, int month, int date, int hours, int minutes)`

This constructor initializes a newly created `Date` object so that it represents the instant at the start of the minute specified by the `year`, `month`, `date`, `hours`, and `minutes` arguments, in the local time zone. Thus, it has the same effect as the constructor call ([§21.3.5](#)):

```
Date(year, month, date, hours, minutes, 0)
```

21.3.5 `public Date(int year, int month, int date, int hours, int minutes, int seconds)`

This constructor initializes a newly created `Date` object so that it represents the instant at the start of the second specified by the `year`, `month`, `date`, `hours`, `minutes`, and `seconds` arguments, in the local time zone.

21.3.6 `public Date(String s)` `throws IllegalArgumentException`

This constructor initializes a newly created `Date` object so that it represents the date and time indicated by the string `s`, which is interpreted as if by the `parse` method ([§21.3.31](#)).

21.3.7 `public String toString()`

This `Date` object is converted to a `String` of the form:

```
"dow mon dd hh:mm:ss zzz yyyy"
```

where:

- `dow` is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat).

- `mon` is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).
- `dd` is the day of the month (01 through 31), as two decimal digits.
- `hh` is the hour of the day (00 through 23), as two decimal digits.
- `mm` is the minute within the hour (00 through 59), as two decimal digits.
- `ss` is the second within the minute (00 through 61), as two decimal digits.
- `zzz` is the time zone (and may reflect daylight saving time). Standard time zone abbreviations include those recognized by the method `parse` (§21.3.31). If time zone information is not available, then `zzz` is empty—that is, it consists of no characters at all.
- `yyyy` is the year, as four decimal digits.

See also methods `toLocaleString` (§21.3.27) and `toGMTString` (§21.3.28).

Overrides the `toString` method of `Object` (§20.1.2).

21.3.8 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `Date` object that represents the same point in time, to the millisecond, as this `Date` object. Thus two `Date` objects are equal if and only if the `getTime` method (§21.3.23) returns the same `long` value from both.

Overrides the `equals` method of `Object` (§20.1.3).

21.3.9 `public int hashCode()`

The result is the exclusive OR of the two halves of the primitive `long` value returned by the `getTime` method (§21.3.23). That is, the hash code is the value of the expression:

```
(int) (this.getTime() ^ (this.getTime() >>> 32))
```

Overrides the `hashCode` method of `Object` (§20.1.4).

21.3.10 `public int getYear()`

The returned value is the result of subtracting 1900 from the year that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.11 `public void setYear(int year)`

This `Date` object is modified so that it represents a point in time within the specified year, with the month, date, hour, minute, and second the same as before, as interpreted in the local time zone. (Of course, if the date was February 29, for example, and the year is set to a non-leap year, then the new date will be treated as if it were on March 1.)

21.3.12 `public int getMonth()`

The returned value is a number (0 through 11) representing the month that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.13 `public void setMonth(int month)`

This `Date` object is modified so that it represents a point in time within the specified month, with the year, date, hour, minute, and second the same as before, as interpreted in the local time zone. If the date was October 31, for example, and the month is set to June, then the new date will be treated as if it were on July 1, because June has only 30 days.

21.3.14 `public int getDate()`

The returned value is a number (1 through 31) representing day of the month that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.15 `public void setDate(int date)`

This `Date` object is modified so that it represents a point in time within the specified day of the month, with the year, month, hour, minute, and second the same as before, as interpreted in the local time zone. If the date was April 30, for example, and the date is set to 31, then it will be treated as if it were on May 1, because April has only 30 days.

21.3.16 `public int getDay()`

The returned value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.17 `public int getHours()`

The returned value is a number (0 through 23) representing the hour within the day that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.18 `public void setHours(int hours)`

This `Date` object is modified so that it represents a point in time within the specified hour of the day, with the year, month, date, minute, and second the same as before, as interpreted in the local time zone.

21.3.19 `public int getMinutes()`

The returned value is a number (0 through 59) representing the minute within the hour that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.20 `public void setMinutes(int minutes)`

This `Date` object is modified so that it represents a point in time within the specified minute of the hour, with the year, month, date, hour, and second the same as before, as interpreted in the local time zone.

21.3.21 `public int getSeconds()`

The returned value is a number (0 through 61) representing the second within the minute that contains or begins with the instant in time represented by this `Date` object, as interpreted in the local time zone.

21.3.22 `public void setSeconds(int seconds)`

This `Date` object is modified so that it represents a point in time within the specified second of the minute, with the year, month, date, hour, and minute the same as before, as interpreted in the local time zone.

21.3.23 `public long getTime()`

This method returns the time represented by this `Date` object, represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970).

21.3.24 `public void setTime(long time)`

This `Date` object is modified so that it represents a point in time that is `time` milliseconds after the epoch (00:00:00 GMT on January 1, 1970).

21.3.25 `public boolean before(Date when)`

The result is `true` if and only if the instant represented by this `Date` object is strictly earlier than the instant represented by `when`.

21.3.26 `public boolean after(Date when)`

The result is `true` if and only if the instant represented by this `Date` object is strictly later than the instant represented by `when`.

21.3.27 `public String toLocaleString()`

This `Date` object is converted to a `String` of an implementation-dependent form. The general intent is that the form should be familiar to the user of the Java application, wherever it may happen to be running. The intent is comparable to that of the `%c` format supported by the `strftime` function of ISO C.

See also methods `toString` ([§21.3.7](#)) and `toGMTString` ([§21.3.28](#)).

21.3.28 `public String toGMTString()`

This `Date` object is converted to a `String` of length 23 or 24 of the form:

`"d mon yyyy hh:mm:ss GMT"`

where:

- `d` is the day of the month (1 through 31), as one or two decimal digits.

- *mon* is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).
- *yyyy* is the year, as four decimal digits.
- *hh* is the hour of the day (00 through 23), as two decimal digits.
- *mm* is the minute within the hour (00 through 59), as two decimal digits.
- *ss* is the second within the minute (00 through 61), as two decimal digits.
- *GMT* is exactly the ASCII letters "GMT" to indicate Greenwich Mean Time.

The result does not depend on the local time zone.

See also methods `toString` (§21.3.7) and `toLocaleString` (§21.3.27).

21.3.29 `public int getTimezoneOffset()`

This method returns the offset, measured in minutes, for the local time zone relative to UTC that is appropriate for the time represented by this `Date` object.

For example, in Massachusetts, five time zones west of Greenwich:

```
new Date(96, 1, 14).getTimezoneOffset() returns 300
```

because on February 14, 1996, standard time (Eastern Standard Time) is in use, which is offset five hours from UTC; but:

```
new Date(96, 5, 1).getTimezoneOffset() returns 240
```

because on May 1, 1996, daylight saving time (Eastern Daylight Time) is in use, which is offset only four hours from UTC.

This method produces the same result as if it computed:

```
(this.getTime() - UTC(this.getYear(),
                      this.getMonth(),
                      this.getDate(),
                      this.getHours(),
                      this.getMinutes(),
                      this.getSeconds())) / (60 * 1000)
```

21.3.30 `public static long UTC(int year, int month, int date,
int hours, int minutes, int seconds)`

The arguments are interpreted as a year, month, day of the month, hour of the day, minute within the hour, and second within the minute, exactly as for the `Date` constructor of six arguments (§21.3.5), except that the arguments are interpreted relative to UTC rather than to the local time zone. The time indicated is returned represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970).

21.3.31 `public static long parse(String s)`

throws `IllegalArgumentException`

An attempt is made to interpret the string `s` as a representation of a date and time. If the attempt is successful, the time indicated is returned represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970). If the attempt fails, an `IllegalArgumentException` is thrown.

The string `s` is processed from left to right, looking for data of interest.

Any material in `s` that is within the ASCII parenthesis characters (and) is ignored. Parentheses may be nested. Otherwise, the only characters permitted within `s` are these ASCII characters:

```
abcdefghijklmnopqrstuvwxyz  
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
0123456789,+ - : /
```

and whitespace characters ([§20.5.19](#)).

A consecutive sequence of decimal digits is treated as a decimal number:

- If a number is preceded by + or – and a year has already been recognized, then the number is a time-zone offset. If the number is less than 24, it is an offset measured in hours. Otherwise, it is regarded as an offset in minutes, expressed in 24-hour time format without punctuation. A preceding + means an eastward offset and a preceding – means a westward offset. Time zone offsets are always relative to UTC (Greenwich). Thus, for example, –5 occurring in the string would mean "five hours west of Greenwich" and +0430 would mean "four hours and thirty minutes east of Greenwich." It is permitted for the string to specify GMT, UT, or UTC redundantly—for example, GMT–5 or utc+0430.
- If a number is greater than 70, it is regarded as a year number. It must be followed by a space, comma, slash, or end of string. If it is greater than 1900, then 1900 is subtracted from it.
- If the number is followed by a colon, it is regarded as an hour, unless an hour has already been recognized, in which case it is regarded as a minute.
- If the number is followed by a slash, it is regarded as a month (it is decreased by 1 to produce a number in the range 0 to 11), unless a month has already been recognized, in which case it is regarded as a day of the month.
- If the number is followed by whitespace, a comma, a hyphen, or end of string, then if an hour has been recognized but not a minute, it is regarded as a minute; otherwise, if a minute has been recognized but not a second, it is regarded as a second; otherwise, it is regarded as a day of the month.

A consecutive sequence of letters is regarded as a word and treated as follows:

- A word that matches AM, ignoring case, is ignored (but the parse fails if an hour has not been recognized or is less than 1 or greater than 12).
- A word that matches PM, ignoring case, adds 12 to the hour (but the parse fails if an hour has not been recognized or is less than 1 or greater than 12).
- Any word that matches any prefix of SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, or SATURDAY, ignoring case, is ignored. For example, sat, Friday, TUE, and Thurs are ignored.
- Otherwise, any word that matches any prefix of JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, or DECEMBER, ignoring case, and considering them in the order given here, is recognized as specifying a month and is converted to a number (0 to 11). For example, aug, Sept, april, and NOV are recognized as months. So

is `Ma`, which is recognized as `MARCH`, not `MAY`.

- Any word that matches `GMT`, `UT`, or `UTC`, ignoring case, is treated as referring to `UTC`.
- Any word that matches `EST`, `CST`, `MST`, or `PST`, ignoring case, is recognized as referring to the time zone in North America that is five, six, seven, or eight hours west of Greenwich, respectively. Any word that matches `EDT`, `CDT`, `MDT`, or `PDT`, ignoring case, is recognized as referring to the same time zone, respectively, during daylight saving time. (In the future, this method may be upgraded to recognize other time zone designations.)

Once the entire string `s` has been scanned, it is converted to a time result in one of two ways. If a time zone or time-zone offset has been recognized, then the year, month, day of month, hour, minute, and second are interpreted in `UTC` ([§21.3.30](#)) and then the time-zone offset is applied. Otherwise, the year, month, day of month, hour, minute, and second are interpreted in the local time zone.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.4 The Class java.util.Dictionary

A `Dictionary` is an object that associates *elements* with *keys*. Every key and every element is an object. In any one `Dictionary`, every key is associated at most one element. Given a `Dictionary` and a key, the associated element can be looked up.

```
public abstract class Dictionary {
    abstract public int size();
    abstract public boolean isEmpty();
    abstract public Object get(Object key)
        throws NullPointerException;
    abstract public Object put(Object key, Object element)
        throws NullPointerException;
    abstract public Object remove(Object key)
        throws NullPointerException;
    abstract public Enumeration keys();
    abstract public Enumeration elements();
}
```

As a rule, the `equals` method (§20.1.3) should be used by implementations of the class `Dictionary` to decide whether two keys are the same.

21.4.1 abstract public int size()

The general contract for the `size` method is that it returns the number of entries (distinct keys) in this dictionary.

21.4.2 abstract public boolean isEmpty()

The general contract for the `isEmpty` method is that the result is `true` if and only if this dictionary contains no entries.

21.4.3 abstract public Object get(Object key) throws NullPointerException

The general contract for the `get` method is that if this dictionary contains an entry for the specified `key`, the associated element is returned; otherwise, `null` is returned.

If the `key` is `null`, a `NullPointerException` is thrown.

21.4.4 abstract public Object put(Object key, Object element) throws NullPointerException

The general contract for the `put` method is that it adds an entry to this dictionary.

If this dictionary already contains an entry for the specified `key`, the element already in this dictionary for that `key` is returned, after modifying the entry to contain the new `element`.

If this dictionary does not already have an entry for the specified `key`, an entry is created for the specified `key` and `element`, and `null` is returned.

If the `key` or the `element` is `null`, a `NullPointerException` is thrown.

21.4.5 `abstract public Object remove(Object key)`
throws `NullPointerException`

The general contract for the `remove` method is that it removes an entry from this dictionary.

If this dictionary contains an entry for the specified `key`, the element in this dictionary for that `key` is returned, after removing the entry from this dictionary.

If this dictionary does not already have an entry for the specified `key`, `null` is returned.

If the `key` is `null`, a `NullPointerException` is thrown.

21.4.6 `abstract public Enumeration keys()`

The general contract for the `keys` method is that an `Enumeration` ([§21.1](#)) is returned that will generate all the keys for which this dictionary contains entries.

21.4.7 `abstract public Enumeration elements()`

The general contract for the `elements` method is that an `Enumeration` ([§21.1](#)) is returned that will generate all the elements contained in entries in this dictionary.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.5 The Class java.util.Hashtable

... never did they seem to have new experiences in common . . . and the things they had for dissection college, contemporary personality, and the like--they had hashed and rehashed for many a frugal conversational meal.

--F. Scott Fitzgerald, *This Side of Paradise* (1920)

The class `Hashtable` implements the abstract class `Dictionary` (§21.4), with some additional functionality.

```
public class Hashtable extends Dictionary implements Cloneable {
    public Hashtable(int initialCapacity, float loadFactor);
    public Hashtable(int initialCapacity);
    public Hashtable();
    public String toString();
    public Object clone();
    public int size();
    public boolean isEmpty();
    public Object get(Object key)
        throws NullPointerException;
    public Object put(Object key, Object value)
        throws NullPointerException;
    public Object remove(Object key)
        throws NullPointerException;
    public Enumeration keys();
    public Enumeration elements();
    public boolean contains(Object value);
    public boolean containsKey(Object key);
    protected void rehash();
    public void clear();
}
```

A `Hashtable` has two parameters that affect its efficiency: its *capacity* and its *load factor*. The load factor should be between 0.0 and 1.0. When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased, using the `rehash` method. Larger load factors use memory more efficiently at the expense of larger expected time per lookup. If many entries are to be made in a `Hashtable`, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

21.5.1 `public Hashtable(int initialCapacity, float loadFactor)`

This constructor initializes a newly created `Hashtable` object so that its capacity is `initialCapacity` and its load factor is `loadFactor`. Initially, there are no entries in the table.

21.5.2 `public Hashtable(int initialCapacity)`

This constructor initializes a newly created `Hashtable` object so that its capacity is `initialCapacity` and its load factor is 0.75. Initially, there are no entries in the table.

21.5.3 `public Hashtable()`

This constructor initializes a newly created `Hashtable` object so that its load factor is 0.75. Initially, there are no entries in the table.

21.5.4 `public String toString()`

This `Hashtable` is represented in string form as a set of entries, enclosed in braces and separated by the ASCII characters ", " (comma and space). Each entry is rendered as the key, an equals sign =, and the associated element, where the `toString` method is used to convert the key and element to strings.

Overrides the `toString` method of `Object` ([§21.2.3](#)).

21.5.5 `public Object clone()`

A copy of this `Hashtable` is constructed and returned. All the structure of the hashtable itself is copied, but the keys and elements are not cloned.

Overrides the `clone` method of `Object` ([§21.2.6](#)).

21.5.6 `public int size()`

Implements the `size` method of `Dictionary` ([§21.4.1](#)).

21.5.7 `public boolean isEmpty()`

Implements the `isEmpty` method of `Dictionary` ([§21.4.2](#)).

21.5.8 `public Object get(Object key)`

Implements the `get` method of `Dictionary` ([§21.4.3](#)).

21.5.9 `public Object put(Object key, Object value)`

Implements the `put` method of `Dictionary` ([§21.4.4](#)).

21.5.10 `public Object remove(Object key)`

Implements the `remove` method of `Dictionary` ([§21.4.5](#)).

21.5.11 `public Enumeration keys()`

Implements the `keys` method of `Dictionary` ([§21.4.6](#)).

21.5.12 `public Enumeration elements()`

Implements the `elements` method of `Dictionary` ([§21.4.7](#)).

21.5.13 `public boolean contains(Object value)`

The result is `true` if and only if this `Hashtable` contains at least one entry for which the element is equal to `value`, as determined by the `equals` method ([§20.1.3](#)).

21.5.14 `public boolean containsKey(Object key)`

The result is `true` if and only if this `Hashtable` contains an entry for which the key is equal to `key`, as determined by the `equals` method ([§20.1.3](#)). In other words, this method produces the same result as the expression:

```
get(key) != null
```

21.5.15 `protected void rehash()`

This `Hashtable` is increased in capacity and reorganized internally, in order to accommodate and access its entries more efficiently.

21.5.16 `public void clear()`

The `clear` method removes all entries from this `Hashtable`.

*Twelve sphered tables, by silk seats insphered,
High as the level of a man's breast rear'd
On libbard's paws, upheld the heavy gold
Of cups and goblets, and the store thrice told
Of Ceres' horn, and, in huge vessels, wine
Came from the gloomy tun with merry shine.
Thus loaded with a feast the tables stood . . .
--John Keats, *Lamia* Part II*

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.6 The Class java.util.Properties

A `Properties` table is a kind of `Hashtable` with two functionality extensions and with the restriction that keys and elements must be strings. First, there are methods for reading entries into the table from an input stream and writing all the entries in the table to an output stream. Second, a `Properties` table may refer to another `Properties` table that provides default values. The `getProperty` method is much like the `get` method ([§21.4.3](#)), but if an entry is not found in this table, then the defaults table is searched (and that defaults table may itself refer to another defaults table, and so on, recursively).

```
public class Properties extends Hashtable {
    protected Properties defaults;
    public Properties();
    public Properties(Properties defaults);
    public String getProperty(String key);
    public String getProperty(String key, String defaultValue);
    public Enumeration propertyNames();
    public void load(InputStream in) throws IOException;
    public void save(OutputStream out, String header);
    public void list(PrintStream out);
}
```

21.6.1 `protected Properties defaults;`

If the `defaults` field is not null, it is another `Properties` table that provides default values for this `Properties` table.

21.6.2 `public Properties()`

This constructor initializes a newly created `Properties` table so that it has no defaults table. Initially, there are no entries in the newly created table.

21.6.3 `public Properties(Properties defaults)`

This constructor initializes a newly created `Properties` table so its defaults table is `defaults`. The argument `defaults` may be null, in which case the newly created `Properties` table will not have a defaults table. Initially, there are no entries in the newly created table.

21.6.4 `public String getProperty(String key)`

If there is an entry in this `Properties` table with `key` as its key, the associated element is returned. Otherwise, if this `Properties` table has a defaults table, then whatever its `getProperty` method returns is returned. Otherwise, null is returned.

21.6.5 `public String getProperty(String key,
 String defaultValue)`

If there is an entry in this `Properties` table with `key` as its key, the associated element is returned.

Otherwise, if this `Properties` table has a defaults table, then whatever its `getProperty` method returns is returned. Otherwise, `defaultValue` is returned.

21.6.6 `public Enumeration propertyNames()`

An `Enumeration` (§21.1) is returned that will generate all the keys for which this `Properties` table could supply an associated element. If this `Properties` table has a defaults table (§21.6.1), then keys for which the defaults table has entries are also supplied by the `Enumeration`, and so on, recursively; but no key is supplied by the `Enumeration` more than once.

21.6.7 `public void load(InputStream in) throws IOException`

`Properties` (key and element pairs) are read from the input stream:

```
Runtime.getRuntime().getLocalizedInputStream(in)
```

and added to this `Properties` table. See the `getLocalizedInputStream` method of `Runtime` (§20.16.15).

Every property occupies one line of the input stream. Each line is terminated by a line terminator (`\n` or `\r` or `\r\n`). Lines from the input stream are processed until end of file is reached on the input stream.

A line that contains only whitespace (§20.5.19) or whose first non-whitespace character is an ASCII `#` or `!` is ignored (thus, `#` or `!` indicate comment lines).

Every line other than a blank line or a comment line describes one property to be added to the table (except that if a line ends with `\`, then the following line is treated as a continuation line, as described below). The key consists of all the characters in the line starting with the first non-whitespace character and up to, but not including, the first ASCII `=`, `:`, or whitespace character. Any whitespace after the key is skipped; if the first non-whitespace character after the key is `=` or `:`, then it is ignored and any whitespace characters after it are also skipped. All remaining characters on the line become part of the associated element string. Within the element string (but not the key), the ASCII escape sequences `\t`, `\n`, `\r`, `\\`, `\"`, `\'`, `\` (a backslash and a space), and `\uxxxx` are recognized and converted to single characters. Moreover, if the last character on the line is `\`, then the next line is treated as a continuation of the current line; the `\` and line terminator are simply discarded, and any leading whitespace characters on the continuation line are also discarded and are not part of the element string.

As an example, each of the following four lines specifies the key `"Truth"` and the associated element value `"Beauty"`:

```
Truth Beauty
Truth = Beauty
    Truth:Beauty
Truth          :Beauty
```

As another example, the following three lines specify a single property:


```
fruits          apple, banana, pear, \  
                cantaloupe, watermelon, \  
                kiwi, mango
```

The key is "fruit" and the associated element is:

```
"apple, banana, pear, cantaloupe, watermelon, kiwi, mango"
```

Note that a space appears before each `\` so that a space will appear after each comma in the final result; the `\`, line terminator, and leading whitespace on the continuation line are merely discarded and are *not* replaced by one or more other characters.

As a third example, the line:

```
cheeses
```

specifies that the key is "cheeses" and the associated element is the empty string.

21.6.8 `public void save(OutputStream out, String header)`

All the properties (key and element pairs) in this `Properties` table are written to the output stream:

```
Runtime.getRuntime().getLocalizedOutputStream(out)
```

in a format suitable for loading into a `Properties` table using the `load` method ([§21.6.7](#)). See the `getLocalizedOutputStream` method of `Runtime` ([§20.16.16](#)).

Properties from the defaults table of this `Properties` table (if any) are *not* written out by this method.

If the header argument is not null, then an ASCII # character, the header string, and a newline are first written to the output stream. Thus, the header can serve as an identifying comment.

Next, a comment line is always written, consisting of an ASCII # character, the current date and time (as if produced by the `toString` method of `Date` ([§21.3.7](#)) for the current time), and a newline.

Then every entry in this `Properties` table is written out, one per line. For each entry the key string is written, then an ASCII =, then the associated element string. Each character of the element string is examined to see whether it should be rendered as an escape sequence. The ASCII characters `\`, tab, newline, and carriage return are written as `\\`, `\t`, `\n`, and `\r`, respectively. Characters less than `\u0020` and characters greater than `\u007E` (if necessary, depending on the needs of the localized output stream) are written as `\uxxxx` for the appropriate hexadecimal value `xxxx`. Leading space characters, but not embedded or trailing space characters, are written with a preceding `\`.

21.6.9 `public void list(PrintStream out)`

Properties (key and element pairs) in this `Properties` table are written to the output stream `out` in a possibly abbreviated form that may be more convenient for use in debugging than the output of the `save` method. No header is written, and element values longer than 40 character are truncated to

the first 37 characters, to which the characters ". . ." are appended. Thus, if the names of the keys are not too long, there is a fighting chance that each property will fit into the space of one line of a physical output device.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.7 The Class java.util.Observable

Each instance of class `Observable` maintains a set of "observers" that are notified whenever the `Observable` object changes in some significant way. An observer may be any object that implements interface `Observer` (§21.8).

Note that this notification mechanism is has nothing to do with threads (§20.20) and is completely separate from the `wait` and `notify` mechanism of class `Object` (§20.1).

```
public class Observable {
    public void addObserver(Observer o);
    public void deleteObserver(Observer o);
    public void deleteObservers();
    public int countObservers();
    public void notifyObservers();
    public void notifyObservers(Object arg);
    protected void setChanged();
    protected void clearChanged();
    public boolean hasChanged();
}
```

When an observable object is newly created, its set of observers is empty.

Two observers are considered the same if and only if the `equals` method (§20.1.3) returns `true` for them.

21.7.1 `public void addObserver(Observer o)`

The observer `o` is added to this `Observable` object's set of observers, provided that it is not the same as some observer already in the set.

21.7.2 `public void deleteObserver(Observer o)`

The observer `o` is removed from this `Observable` object's set of observers.

21.7.3 `public void deleteObservers()`

All observers are removed from this `Observable` object's set of observers.

21.7.4 `public int countObservers()`

The number of observers in this `Observable` object's set of observers is returned.

21.7.5 `public void notifyObservers()`

If this `Observable` object has been marked as changed, this method causes all observers to be notified with `null` as the second argument; in other words, this method is equivalent to:


```
notifyObservers (null)
```

21.7.6 `public void notifyObservers(Object arg)`

If this `Observable` object has been marked as changed ([§21.7.9](#)), this method causes all observers to be notified with `arg` as the second argument. An observer is notified by calling its `update` method ([§21.8.1](#)) on two arguments: this `Observable` object and `arg`. The mark on this object is then cleared ([§21.7.8](#)).

21.7.7 `protected void setChanged()`

This `Observable` object is marked as having been changed; the `hasChanged` method will now return `true`.

21.7.8 `protected void clearChanged()`

This `Observable` object is marked as not having been changed; the `hasChanged` method will now return `false`.

21.7.9 `public boolean hasChanged()`

The result is `true` if and only if the `setChanged` method has been called for this `Observable` object more recently than either the `clearChanged` method or the `notifyObservers` method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.8 The Interface java.util.Observer

A class should implement the `Observer` interface if it is to be notified whenever an `Observable` object has been changed. See the `Observable` class ([§21.7](#)) for a discussion of how `Observer` objects are notified.

```
public interface Observer {  
    public void update(Observable o, Object arg);  
}
```

21.8.1 `public void update(Observable o, Object arg)`

When an `Observable` object has been changed and its `notifyObservers` method ([§21.7.6](#)) is called, every `Observer` object in its set of `observers` is notified by invoking its `update` method, passing it two arguments: the `Observable` object and another argument specified by the call to the `notifyObservers` method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.9 The Class `java.util.Random`

*Oh, many a shaft at random sent
Finds mark the archer little meant!
And many a word at random spoken
May soothe, or wound, a heart that's broken!*
--Sir Walter Scott, *The Lady of the Lake*, Canto V, stanza 18

Each instance of class `Random` serves as a separate, independent pseudorandom generator of primitive values.

```
public class Random {
    protected long seed;
    protected double nextNextGaussian;
    protected boolean haveNextNextGaussian = false;
    public Random();
    public Random(long seed);
    protected int next(int bits);
    public void setSeed(long seed);
    public int nextInt();
    public long nextLong();
    public float nextFloat();
    public double nextDouble();
    public double nextGaussian();
}
```

If two `Random` objects are created with the same seed and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers in all Java implementations. In order to guarantee this property, particular algorithms are specified for the class `Random`. Java implementations must use all the algorithms shown here for the class `Random`, for the sake of absolute portability of Java code. However, subclasses of class `Random` are permitted use other algorithms, so long as they adhere to the general contracts for all the methods.

The algorithms implemented by class `Random` use three state variables, which are `protected`. They also use a `protected` utility method that on each invocation can supply up to up to 32 pseudorandomly generated bits.

21.9.1 `protected long seed;`

A variable used by method `next` (§21.9.7) to hold the state of the pseudorandom number generator.

21.9.2 `protected double nextNextGaussian;`

A variable used by method `nextGaussian` (§21.9.12) to hold a precomputed value to be delivered by that method the next time it is called.

21.9.3 `protected boolean haveNextNextGaussian = false;`

A variable used by method `nextGaussian` (§21.9.12) to keep track of whether it has

precomputed and stashed away the next value to be delivered by that method.

21.9.4 `public Random()`

This constructor initializes a newly created `Random` number generator by using the current time of day (§20.18.6) as a seed.

```
public Random() { this(System.currentTimeMillis()); }
```

21.9.5 `public Random(long seed)`

This constructor initializes a newly created `Random` number generator by using the argument `seed` as a seed.

```
public Random(long seed) { setSeed(seed); }
```

21.9.6 `public void setSeed(long seed)`

The general contract of `setSeed` is that it alters the state of this random number generator object so as to be in exactly the same state as if it had just been created with the argument `seed` as a seed.

The method `setSeed` is implemented by class `Random` as follows:

```
synchronized public void setSeed(long seed) {  
    this.seed = (seed ^ 0x5DEECE66DL) & ((1L << 48) - 1);  
    haveNextNextGaussian = false;  
}
```

The implementation of `setSeed` by class `Random` happens to use only 48 bits of the given seed. In general, however, an overriding method may use all 64 bits of the long argument as a seed value.

[In certain early versions of Java, the `setSeed` method failed to reset the value of `haveNextNextGaussian` to `false`; this flaw could lead to failure to produce repeatable behavior.]

21.9.7 `protected int next(int bits)`

The general contract of `next` is that it returns an `int` value and if the argument `bits` is between 1 and 32 (inclusive), then that many low-order bits of the returned value will be (approximately) independently chosen bit values, each of which is (approximately) equally likely to be 0 or 1.

The method `next` is implemented by class `Random` as follows:

```
synchronized protected int next(int bits) {  
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
```



```

        return (int)(seed >>> (48 - bits));
    }

```

This is a linear congruential pseudorandom number generator, as defined by D. H. Lehmer and described by Donald E. Knuth in *The Art of Computer Programming*, Volume 2: *Seminumerical Algorithms*, section 3.2.1.

21.9.8 `public int nextInt()`

The general contract of `nextInt` is that one `int` value is pseudorandomly generated and returned. All {ewc msdnrd, EWGraphic, LNG9y 0 /a "langref.BMP"}possible `int` values are produced with (approximately) equal probability.

The method `setSeed` is implemented by class `Random` as follows:

```

public int nextInt() { return next(32); }

```

21.9.9 `public long nextLong()`

The general contract of `nextLong` is that one `long` value is pseudorandomly generated and returned. All {ewc msdnrd, EWGraphic, LNG9y 1 /a "langref.BMP"}possible `long` values are produced with (approximately) equal probability.

The method `setSeed` is implemented by class `Random` as follows:

```

public long nextLong() {
    return ((long)next(32) << 32) + next(32);
}

```

21.9.10 `public float nextFloat()`

The general contract of `nextFloat` is that one `float` value, chosen (approximately) uniformly from the range `0.0f` (inclusive) to `1.0f` (exclusive), is pseudorandomly generated and returned. All {ewc msdnrd, EWGraphic, LNG9y 2 /a "langref.BMP"}possible `float` values of the form {ewc msdnrd, EWGraphic, LNG9y 3 /a "langref.BMP"}, where m is a positive integer less than {ewc msdnrd, EWGraphic, LNG9y 4 /a "langref.BMP"}, are produced with (approximately) equal probability.

The method `setSeed` is implemented by class `Random` as follows:

```

public float nextFloat() {
    return next(24) / ((float)(1 << 24));
}

```

The hedge "approximately" is used in the foregoing description only because the `next` method is

only approximately an unbiased source of independently chosen bits. If it were a perfect source or randomly chosen bits, then the algorithm shown would choose `float` values from the stated range with perfect uniformity.

[In early versions of Java, the result was incorrectly calculated as:

```
return next(30) / ((float) (1 << 30));
```

This might seem to be equivalent, if not better, but in fact it introduced a slight nonuniformity because of the bias in the rounding of floating-point numbers: it was slightly more likely that the low-order bit of the significand would be 0 than that it would be 1.]

21.9.11 `public double nextDouble()`

The general contract of `nextDouble` is that one `double` value, chosen (approximately) uniformly from the range 0.0d (inclusive) to 1.0d (exclusive), is pseudorandomly generated and returned. All possible `float` values of the form $\frac{m}{2^m}$, where m is a positive integer less than 32, are produced with (approximately) equal probability.

The method `setSeed` is implemented by class `Random` as follows:

```
public double nextDouble() {
    return (((long)next(26) << 27) + next(27))
           / (double) (1L << 53);
}
```

The hedge "approximately" is used in the foregoing description only because the `next` method is only approximately an unbiased source of independently chosen bits. If it were a perfect source or randomly chosen bits, then the algorithm shown would choose `double` values from the stated range with perfect uniformity.

[In early versions of Java, the result was incorrectly calculated as:

```
return (((long)next(27) << 27) + next(27))
       / (double) (1L << 54);
```

This might seem to be equivalent, if not better, but in fact it introduced a large nonuniformity because of the bias in the rounding of floating-point numbers: it was three times as likely that the low-order bit of the significand would be 0 than that it would be 1! This nonuniformity probably doesn't matter much in practice, but we strive for perfection.]

21.9.12 `public double nextGaussian()`

The general contract of `nextGaussian` is that one `double` value, chosen from (approximately) the usual normal distribution with mean 0.0 and standard deviation 1.0, is pseudorandomly generated and returned.

The method `setSeed` is implemented by class `Random` as follows:

```
synchronized public double nextGaussian() {
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
            v2 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
            s = v1 * v1 + v2 * v2;
        } while (s >= 1);
        double norm = Math.sqrt(-2 * Math.log(s)/s);
        nextNextGaussian = v2 * norm;
        haveNextNextGaussian = true;
        return v1 * norm;
    }
}
```

This uses the *polar method* of G. E. P. Box, M. E. Muller, and G. Marsaglia, as described by Donald E. Knuth in *The Art of Computer Programming*, Volume 2: *Seminumerical Algorithms*, section 3.4.1, subsection C, algorithm P. Note that it generates two independent values at the cost of only one call to `Math.log` and one call to `Math.sqrt`.

... who can tell what may be the event? ...

The mind of the multitude is left at random ...

--Thomas Paine, *Common Sense* (1776), Appendix A

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.10 The Class `java.util.StringTokenizer`

The `StringTokenizer` class provides a way to break a `String` into tokens. The tokenizing method used by this class is much simpler than the one used by the class `java.io.StreamTokenizer`. For example, a `StringTokenizer` does not distinguish among identifiers, numbers, and quoted strings; moreover, it does not recognize and skip comments.

A `StringTokenizer` can serve as an Enumeration ([§21.1](#)).

```
public class StringTokenizer implements Enumeration {
    public StringTokenizer(String str, String delim,
        boolean returnTokens);
    public StringTokenizer(String str, String delim);
    public StringTokenizer(String str);
    public boolean hasMoreTokens();
    public String nextToken();
    public String nextToken(String delim);
    public boolean hasMoreElements();
    public Object nextElement();
    public int countTokens();
}
```

A `StringTokenizer` simply divides characters into classes: delimiters and other characters. The tokenizer behaves in one of two ways, depending on whether it was created with `returnTokens` having the value `true` or `false`.

If `returnTokens` is `false`, delimiter characters merely serve to separate tokens of interest. A token is thus a maximal sequence of consecutive characters that are not delimiters.

If `returnTokens` is `true`, delimiter characters are themselves considered to be tokens of interest. A token is thus either one delimiter character or a maximal sequence of consecutive characters that are not delimiters.

A `StringTokenizer` internally maintains a current position within the `String` to be tokenized. Some operations advance this current position past the characters processed.

A token is returned by taking a substring ([§20.12.32](#)) of the string that was used to create the `StringTokenizer`.

21.10.1 `public StringTokenizer(String str, String delim,`
 `boolean returnTokens)`

This constructor initializes a newly created `StringTokenizer` so that it will recognize tokens within the given string `str`. All characters in the string `delim` will be considered delimiters. The argument `returnTokens` specifies whether delimiter characters themselves are to be considered tokens.

21.10.2 `public StringTokenizer(String str, String delim)`

This constructor initializes a newly created `StringTokenizer` so that it will recognize tokens within the given string `str`. All characters in the string `delim` will be considered delimiters. Delimiter characters themselves will not be treated as tokens.

21.10.3 `public StringTokenizer(String str)`

This constructor initializes a newly created `StringTokenizer` so that it will recognize tokens within the given string `str`. All whitespace characters ([§20.5.19](#)) will be considered delimiters. Delimiter characters themselves will not be treated as tokens.

21.10.4 `public boolean hasMoreTokens()`

The result is `true` if and only if there is at least one token in the string after the current position. If this method returns `true`, then a subsequent call to `nextToken` with no argument will successfully return a token.

21.10.5 `public String nextToken()`

The next token in the string after the current position is returned. The current position is advanced beyond the recognized token.

21.10.6 `public String nextToken(String delim)`

First, the set of characters considered to be delimiters by this `StringTokenizer` is changed to be the characters in the string `delim`. Then the next token in the string after the current position is returned. The current position is advanced beyond the recognized token.

21.10.7 `public boolean hasMoreElements()`

This method has exactly the same behavior as `hasMoreTokens` ([§21.10.4](#)). It is provided so that a `StringTokenizer` can serve as an `Enumeration` ([§21.1](#)).

21.10.8 `public Object nextElement()`

This method has exactly the same behavior as `nextToken` ([§21.10.5](#)). It is provided so that a `StringTokenizer` can serve as an `Enumeration` ([§21.1](#)).

21.10.9 `public int countTokens()`

The result is the number of tokens in the string after the current position, using the current set of delimiter characters. The current position is not advanced.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.11 The Class java.util.Vector

A `Vector`, like an array, contains items that can be accessed using an integer index. However, the size of a `Vector` can grow and shrink as needed to accommodate adding and removing items after the `Vector` has been created.

```
public class Vector implements Cloneable {
    protected Object[] elementData;
    protected int elementCount;
    protected int capacityIncrement;
    public Vector(int initialCapacity, int capacityIncrement);
    public Vector(int initialCapacity);
    public Vector();
    public final String toString();
    public Object clone();
    public final Object elementAt(int index)
        throws IndexOutOfBoundsException;
    public final void setElementAt(Object obj, int index)
        throws IndexOutOfBoundsException;
    public final Object firstElement()
        throws NoSuchElementException;
    public final Object lastElement()
        throws NoSuchElementException;
    public final void addElement(Object obj);
    public final void insertElementAt(Object obj, int index)
        throws IndexOutOfBoundsException;
    public final boolean removeElement(Object obj);
    public final void removeElementAt(int index)
        throws IndexOutOfBoundsException;
    public final void removeAllElements();
    public final boolean isEmpty();
    public final int size();
    public final void setSize(int newSize);
    public final int capacity();
    public final void ensureCapacity(int minCapacity);
    public final void trimToSize();
    public final void copyInto(Object anArray[])
        throws IndexOutOfBoundsException;
    public final Enumeration elements();
    public final boolean contains(Object elem);
    public final int indexOf(Object elem);
    public final int indexOf(Object elem, int index)
        throws IndexOutOfBoundsException;
    public final int lastIndexOf(Object elem);
    public final int lastIndexOf(Object elem, int index)
        throws IndexOutOfBoundsException;
}
```

21.11.1 `protected Object[] elementData;`

Internally, a `Vector` keeps its elements in an array that is at least large enough to contain all the elements.

21.11.2 `protected int elementCount;`

This field holds the number of items currently in this `Vector` object. Components `elementData[0]` through `elementData[elementCount-1]` are the actual items.

21.11.3 `protected int capacityIncrement;`

When the method `ensureCapacity` (§21.11.22) must increase the size of the data array in the field `elementData` (by creating a new array), it increases the size by at least the amount in `capacityIncrement`; but if `capacityIncrement` is zero, then it at least doubles the size of the data array.

21.11.4 `public Vector(int initialCapacity, int capacityIncrement)`

This constructor initializes a newly created `Vector` so that its internal data array has size `initialCapacity` and its standard capacity increment is the value of `capacityIncrement`. Initially, the `Vector` contains no items.

21.11.5 `public Vector(int initialCapacity)`

This constructor initializes a newly created `Vector` so that its internal data array has size `initialCapacity` and its standard capacity increment is zero. Initially, the `Vector` contains no items.

21.11.6 `public Vector()`

This constructor initializes a newly created `Vector` so that its internal data array has size 10 and its standard capacity increment is zero. Initially the `Vector` contains no items.

21.11.7 `public final String toString()`

This `Vector` is represented in string form as a list of its items, enclosed in ASCII square brackets and separated by the ASCII characters ", " (comma and space). The `toString` method is used to convert the items to strings; a null reference is rendered as the string "null".

The example fragment:

```
Vector v = new Vector();
v.addElement("Canberra");
v.addElement("Cancun");
v.addElement("Canandaigua");
System.out.println(v.toString());
```

produces the output:

```
[Canberra, Cancun, Canandaigua]
```


Overrides the `toString` method of `Object` ([§20.1.2](#)).

21.11.8 `public Object clone()`

A copy of this `Vector` is constructed and returned. The copy will contains a reference to a clone of the internal data array, not a reference to the original internal data array of this `Vector`.

Overrides the `clone` method of `Object` ([§20.1.5](#)).

21.11.9 `public final Object elementAt(int index)`

throws `IndexOutOfBoundsException`

The item of this `Vector` with the specified `index` is returned.

If the `index` is negative or not less than the current size of this `Vector`, an `IndexOutOfBoundsException` is thrown.

21.11.10 `public final void setElementAt(Object obj, int index)`

throws `IndexOutOfBoundsException`

The item of this `Vector` with the specified `index` is replaced with `obj`, so that `obj` is now the item at the specified `index` within this `Vector`.

If the `index` is negative or not less than the current size of this `Vector`, an `IndexOutOfBoundsException` is thrown.

21.11.11 `public final Object firstElement()`

throws `NoSuchElementException`

If this `Vector` is empty, a `NoSuchElementException` is thrown. Otherwise, the first item (the item at index 0) is returned.

21.11.12 `public final Object lastElement()`

throws `NoSuchElementException`

If this `Vector` is empty, a `NoSuchElementException` is thrown. Otherwise, the last item (the item at index `size() - 1`) is returned.

21.11.13 `public final void addElement(Object obj)`

The size of this `Vector` is increased by 1 and `obj` becomes the new last item.

21.11.14 `public final void insertElementAt(Object obj, int index)`

throws `IndexOutOfBoundsException`

The size of this `Vector` is increased by 1 and `obj` becomes the new item at the specified `index`. Any item in this `Vector` that was previously at index `k` is first moved to index `k+1` if and only if `k` is not less than `index`.

21.11.15 `public final boolean removeElement(Object obj)`

If this `Vector` contains an occurrence of `obj`, then the first (lowest-indexed) such occurrence is removed, as if by the method `removeElementAt` (§21.11.16), and `true` is returned. If this `Vector` contains no occurrence of `obj`, this `Vector` is not modified and `false` is returned.

21.11.16 `public final void removeElementAt(int index)`
throws `IndexOutOfBoundsException`

The size of this `Vector` is decreased by 1 and the item at the specified `index` is removed from this `Vector`. Any item in this `Vector` that was previously at index `k` is first moved to index `k-1` if and only if `k` is greater than `index`.

21.11.17 `public final void removeAllElements()`

All elements are removed from this `Vector`, making it empty.

21.11.18 `public final boolean isEmpty()`

The result is `true` if and only if this `Vector` is empty, that is, its size is zero.

21.11.19 `public final int size()`

The size of this `Vector` (the number of items it currently contains) is returned.

21.11.20 `public final void setSize(int newSize)`

The size of this `Vector` is changed to `newSize`. If the new size is smaller than the old size, then items are removed from the end and discarded. If the new size is larger than the old size, then the new items are set to `null`.

21.11.21 `public final int capacity()`

The current capacity of this `Vector` (the length of its internal data array, kept in the field `elementData`) is returned.

21.11.22 `public final void ensureCapacity(int minCapacity)`

If the current capacity of this `Vector` is less than `minCapacity`, then its capacity is increased by replacing its internal data array, kept in the field `elementData` (§21.11.1), with a larger one. The size of the new data array will be the old size plus `capacityIncrement` (§21.11.3), unless the value of `capacityIncrement` is nonpositive, in which case the new capacity will be twice the old capacity; but if this new size is still smaller than `minCapacity`, then the new capacity will be `minCapacity`.

21.11.23 `public final void trimToSize()`

If the capacity of this `Vector` is larger than its current size (§21.11.19), then the capacity is changed to equal the size by replacing its internal data array, kept in the field `elementData`, with a

smaller one.

21.11.24 `public final void copyInto(Object anArray[])`
throws `IndexOutOfBoundsException`

All the items in this `Vector` are copied into the array `anArray`. The item at index `k` in this `Vector` is copied into component `k` of `anArray`. If the length of `anArray` is smaller than the size of this `Vector`, an `IndexOutOfBoundsException` is thrown.

21.11.25 `public final Enumeration elements()`

An `Enumeration` (§21.1) is returned that will generate all items in this `Vector`. The first item generated is the item at index 0, then the item at index 1, and so on.

21.11.26 `public final boolean contains(Object elem)`

The result is `true` if and only if some item in this `Vector` is the same as `elem`, as determined by the `equals` method (§20.1.3).

21.11.27 `public final int indexOf(Object elem)`

If an item equal to `elem` is in this `Vector`, then the index of the first such occurrence is returned, that is, the smallest value `k` such that:

```
elem.equals(elementData[k])
```

is `true`. If no such item occurs in this `Vector`, then `-1` is returned.

21.11.28 `public final int indexOf(Object elem, int index)`
throws `IndexOutOfBoundsException`

If an item equal to `elem` is in this `Vector` at position `k` or higher, then the index of the first such occurrence is returned, that is, the smallest value `k` such that:

```
elem.equals(elementData[k]) && (k >= index)
```

is `true`. If no such item occurs in this `Vector`, then `-1` is returned.

21.11.29 `public final int lastIndexOf(Object elem)`

If an item equal to `elem` is in this `Vector`, then the index of the last such occurrence is returned, that is, the largest value `k` such that:

```
elem.equals(elementData[k])
```

is `true`. If no such item occurs in this `Vector`, then `-1` is returned.

21.11.30 `public final int lastIndexOf(Object elem, int index)`
throws `IndexOutOfBoundsException`

If an item equal to `elem` is in this `Vector` at position `k` or lower, then the index of the last such occurrence is returned, that is, the largest value `k` such that:

`elem.equals(elementData[k]) && (k <= index)`

is `true`. If no such item occurs in this `Vector`, then `-1` is returned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

21.12 The Class `java.util.Stack`

*... and from the stack a thin blue wreath of smoke
Curled through the air across the ripening oats ...*
--Oscar Wilde, *Charmides* (1881)

The class `Stack` extends `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top.

```
public class Stack extends Vector {  
    public Object push(Object item);  
    public Object pop() throws EmptyStackException;  
    public Object peek() throws EmptyStackException;  
    public boolean empty();  
    public int search(Object o);  
}
```

When a stack is first created, it contains no items.

21.12.1 `public Object push(Object item)`

The `item` is pushed onto the top of this stack. This has exactly the same effect as:

```
addElement(item)
```

See method `addElement` of `Vector` ([§21.11.13](#)).

21.12.2 `public Object pop() throws EmptyStackException`

If the stack is empty, an `EmptyStackException` is thrown. Otherwise, the topmost item (last item of the `Vector`) is removed and returned.

21.12.3 `public Object peek() throws EmptyStackException`

If the stack is empty, an `EmptyStackException` is thrown. Otherwise, the topmost item (last item of the `Vector`) is returned but not removed.

21.12.4 `public boolean empty()`

The result is `true` if and only if the stack contains no items.

21.12.5 `public int search(Object o)`

If the object `o` occurs as an item in this `Stack`, this method returns the distance from the top of the

stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance 1. The `equals` method ([§20.1.3](#)) is used to compare `o` to the items in this `Stack`.

*. . . And overhead in circling listlessness
The cawing rooks whirl round the frosted stacks . . .*
--Oscar Wilde, *Humanitad* (1881)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.13 The Class java.util.EmptyStackException

A `EmptyStackException` is thrown to indicate an attempt to pop ([§21.12.2](#)) or peek ([§21.12.3](#)) an empty `Stack` object.

```
public class EmptyStackException extends RuntimeException {  
    public EmptyStackException();  
}
```

21.13.1 `public EmptyStackException()`

This constructor initializes a newly created `EmptyStackException` with `null` as its error message string.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


21.14 The Class java.util.NoSuchElementException

A `NoSuchElementException` is thrown to indicate that another element was requested from an `Enumeration` object that has no more elements to supply. See method `nextElement` of interface `Enumeration` ([§21.1.2](#)).

```
public class NoSuchElementException extends RuntimeException {  
    public NoSuchElementException();  
    public NoSuchElementException(String s);  
}
```

21.14.1 `public NoSuchElementException()`

This constructor initializes a newly created `NoSuchElementException` with `null` as its error message string.

21.14.2 `public NoSuchElementException(String s)`

This constructor initializes a newly created `NoSuchElementException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

The Package java.io

Input and output in Java is organized around the concept of streams. A stream is a sequence of items, usually 8-bit bytes, read or written over the course of time.

In the `java.io` package, all input is done through subclasses of the abstract class `InputStream`, and all output is done through subclasses of the abstract class `OutputStream`. The one exception to this rule is the class `RandomAccessFile`, which handles files that allow random access and perhaps intermixed reading and writing of the file.

For an input stream, the source of data might be a file, a `String`, an array of bytes, or bytes written to an output stream (typically by another thread). There are also "filter input streams" that take data from another input stream and transform or augment the data before delivering it as input. For example, a `LineNumberInputStream` passes bytes through verbatim but counts line terminators as they are read.

For an output stream, the sink of data might be a file, an array of bytes, or a buffer to be read as an input stream (typically by another thread). There are also "filter output streams" that transform or augment data before writing it to some other output stream.

An instance of class `File` represents a path name (a string) that might identify a particular file within a file system. Certain operations on the file system, such as renaming and deleting files, are done by this class rather than through streams.

An instance of class `FileDescriptor` represents an abstract indication of a particular file within a file system; such file descriptors are created internally by the Java I/O system.

There are two interfaces, `DataInput` and `DataOutput`, that support the transfer of data other than bytes or characters, such as long integers, floating-point numbers and strings. The class `DataInputStream` implements `DataInput`; the class `DataOutputStream` implements `DataOutput`; and `RandomAccessFile` implements both `DataInput` and `DataOutput`.

The class `StreamTokenizer` provides some simple support for parsing bytes or characters from an input stream into tokens such as identifiers, numbers, and strings, optionally ignoring comments and optionally recognizing or ignoring line terminators.

The hierarchy of classes defined in package `java.io` is as follows. (Classes whose names are shown here in **boldface** are in package `java.io`; the others are in package `java.lang` and are shown here to clarify subclass relationships.)

Object	<u>§20.1</u>
interface DataInput	<u>§22.1</u>
interface DataOutput	<u>§22.2</u>
InputStream	<u>§22.3</u>
FileInputStream	<u>§22.4</u>
PipedInputStream	<u>§22.5</u>
ByteArrayInputStream	<u>§22.6</u>
StringBufferInputStream	<u>§22.7</u>
SequenceInputStream	<u>§22.8</u>
FilterInputStream	<u>§22.9</u>
BufferedInputStream	<u>§22.10</u>
DataInputStream	<u>§22.11</u>
LineNumberInputStream	<u>§22.12</u>

PushBackInputStream	<u>§22.13</u>
StreamTokenizer	<u>§22.14</u>
OutputStream	<u>§22.15</u>
FileOutputStream	<u>§22.16</u>
PipedOutputStream	<u>§22.17</u>
ByteArrayOutputStream	<u>§22.18</u>
FilterOutputStream	<u>§22.19</u>
BufferedOutputStream	<u>§22.20</u>
DataOutputStream	<u>§22.21</u>
PrintStream	<u>§22.22</u>
RandomAccessFile	<u>§22.23</u>
File	<u>§22.24</u>
interface FileNameFilter	<u>§22.25</u>
FileDescriptor	<u>§22.26</u>
Throwable	<u>§20.22</u>
Exception	
IOException	<u>§22.27</u>
EOFException	<u>§22.28</u>
FileNotFoundException	<u>§22.29</u>
InterruptedIOException	<u>§22.30</u>
UTFDataFormatException	<u>§22.31</u>

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.1 The Interface java.io.DataInput

The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types. There is also a facility for reconstructing a `String` from data in Java modified UTF-8 format.

The `DataOutput` interface (§22.2) supports the creation of binary output data suitable for reading back in through the `DataInput` interface.

The `DataInput` interface is implemented by classes `DataInputStream` (§22.11) and `RandomAccessFile` (§22.23).

```
public interface DataInput {
    public void readFully(byte[] b)
        throws IOException, NullPointerException;
    public void readFully(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public int skipBytes(int n) throws IOException;
    public boolean readBoolean() throws IOException;
    public byte readByte() throws IOException;
    public int readUnsignedByte() throws IOException;
    public short readShort() throws IOException;
    public int readUnsignedShort() throws IOException;
    public char readChar() throws IOException;
    public int readInt() throws IOException;
    public long readLong() throws IOException;
    public float readFloat() throws IOException;
    public double readDouble() throws IOException;
    public String readLine() throws IOException;
    public String readUTF() throws IOException;
    public final static String readUTF(DataInput in)
        throws IOException;
}
```

It is generally true of all the reading routines in this interface that if end of file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end of file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the input stream has been closed (§22.3.6).

22.1.1 `public void readFully(byte[] b)`
 throws `IOException`, `NullPointerException`;

The general contract of `readFully(b)` is that it reads some bytes from an input stream and stores them into the buffer array `b`. The number of bytes read is equal to the length of `b`.

This method blocks until one of the following conditions occurs:

- `b.length` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `b.length` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on.

If an exception is thrown from this method, then it may be that some but not all bytes of `b` have been updated with data from the input stream.

22.1.2 `public void readFully(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`,
 `IndexOutOfBoundsException`

The general contract of `readFully(b, off, len)` is that it reads `len` bytes from an input stream.

This method blocks until one of the following conditions occurs:

- `len` bytes of input data are available, in which case a normal return is made.
- End of file is detected, in which case an `EOFException` is thrown.
- An I/O error occurs, in which case an `IOException` other than `EOFException` is thrown.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read. Otherwise, the first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`.

If an exception is thrown from this method, then it may be that some but not all bytes of `b` in positions `off` through `off+len-1` have been updated with data from the input stream.

22.1.3 `public int skipBytes(int n) throws IOException`

The general contract of `skipBytes` is that it makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes. However, it may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. This method never throws an `EOFException`. The actual number of bytes skipped is returned.

22.1.4 `public boolean readBoolean() throws IOException;`

The general contract of `readBoolean` is that it reads one input byte and returns `true` if that byte is nonzero, `false` if that byte is zero.

This method is suitable for reading the byte written by the `writeBoolean` method of interface `DataOutput` ([§22.2.4](#)).

22.1.5 `public byte readByte() throws IOException`

The general contract of `readByte` is that it reads and returns one input byte. The byte is treated as a signed value in the range `-128` through `127`, inclusive.

This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput` ([§22.2.5](#)).

22.1.6 `public int readUnsignedByte() throws IOException`

The general contract of `readUnsignedByte` is that it reads one input byte, zero-extends it to type `int`, and returns the result, which is therefore in the range 0 through 255.

This method is suitable for reading the byte written by the `writeByte` method of interface `DataOutput` ([§22.2.5](#)) if the argument to `writeByte` was intended to be a value in the range 0 through 255.

22.1.7 `public short readShort() throws IOException`

The general contract of `readShort` is that it reads two input bytes and returns a `short` value. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(short)((a << 8) | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput` ([§22.2.6](#)).

22.1.8 `public int readUnsignedShort() throws IOException`

The general contract of `readUnsignedShort` is that it reads two input bytes and returns an `int` value in the range 0 through 65535. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
((a & 0xff) << 8) | (b & 0xff))
```

This method is suitable for reading the bytes written by the `writeShort` method of interface `DataOutput` ([§22.2.6](#)) if the argument to `writeShort` was intended to be a value in the range 0 through 65535.

22.1.9 `public char readChar() throws IOException`

The general contract of `readChar` is that it reads two input bytes and returns a `char` value. Let `a` be the first byte read and `b` be the second byte. The value returned is:

```
(char)((a << 8) | (b & 0xff))
```

This method is suitable for reading bytes written by the `writeChar` method of interface `DataOutput` ([§22.2.7](#)).

22.1.10 `public int readInt() throws IOException`

The general contract of `readInt` is that it reads four input bytes and returns an `int` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, and `d` be the fourth byte. The value returned is:


```
((a & 0xff) << 24) | ((b & 0xff) << 16) |
((c & 0xff) << 8) | (d & 0xff))
```

This method is suitable for reading bytes written by the `writeInt` method of interface `DataOutput` (§22.2.8).

22.1.11 `public long readLong() throws IOException`

The general contract of `readLong` is that it reads eight input bytes and returns a `long` value. Let `a` be the first byte read, `b` be the second byte, `c` be the third byte, `d` be the fourth byte, `e` be the fifth byte, `f` be the sixth byte, `g` be the seventh byte, and `h` be the eighth byte. The value returned is:

```
((long) (a & 0xff) << 56) |
((long) (b & 0xff) << 48) |
((long) (c & 0xff) << 40) |
((long) (d & 0xff) << 32) |
((long) (e & 0xff) << 24) |
((long) (f & 0xff) << 16) |
((long) (g & 0xff) << 8) |
((long) (h & 0xff))
```

This method is suitable for reading bytes written by the `writeLong` method of interface `DataOutput` (§22.2.9).

22.1.12 `public float readFloat() throws IOException`

The general contract of `readFloat` is that it reads four input bytes and returns a `float` value. It does this by first constructing an `int` value in exactly the manner of the `readInt` method (§22.1.10), then converting this `int` value to a `float` in exactly the manner of the method `Float.intBitsToFloat` (§20.9.23).

This method is suitable for reading bytes written by the `writeFloat` method of interface `DataOutput` (§22.2.10).

22.1.13 `public double readDouble() throws IOException`

The general contract of `readDouble` is that it reads eight input bytes and returns a `double` value. It does this by first constructing a `long` value in exactly the manner of the `readLong` method (§22.1.11), then converting this `long` value to a `double` in exactly the manner of the method `Double.longBitsToDouble` (§20.10.22).

This method is suitable for reading bytes written by the `writeDouble` method of interface `DataOutput` (§22.2.11).

22.1.14 `public String readLine() throws IOException`

The general contract of `readLine` is that it reads successive bytes, converting each byte separately into a character, until it encounters a line terminator or end of file; the characters read are then returned as a `String`. Note that because this method processes bytes, it does not support input of the full Unicode character set.

If end of file is encountered before even one byte can be read, then `null` is returned. Otherwise, each byte that is read is converted to type `char` by zero-extension. If the character `'\n'` is encountered, it is discarded and reading ceases. If the character `'\r'` is encountered, it is discarded and, if the following byte converts to the character `'\n'`, then that is discarded also; reading then ceases. If end of file is encountered before either of the characters `'\n'` and `'\r'` is encountered, reading ceases. Once reading has ceased, a `String` is returned that contains all the characters read and not discarded, taken in order. Note that every character in this string will have a value less than `\u0100`, that is, `(char) 256`.

22.1.15 `public String readUTF() throws IOException`

The general contract of `readUTF` is that it reads a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`.

First, two bytes are read and used to construct an unsigned 16-bit integer in exactly the manner of the `readUnsignedShort` method (§22.1.8). This integer value is called the *UTF length* and specifies the number of additional bytes to be read. These bytes are then converted to characters by considering them in groups. The length of each group is computed from the value of the first byte of the group. The byte following a group, if any, is the first byte of the next group.

If the first byte of a group matches the bit pattern `0xxxxxxx` (where `x` means "may be 0 or 1"), then the group consists of just that byte. The byte is zero-extended to form a character.

If the first byte of a group matches the bit pattern `110xxxxx`, then the group consists of that byte `a` and a second byte `b`. If there is no byte `b` (because byte `a` was the last of the bytes to be read), or if byte `b` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
(char) (((a & 0x1F) << 6) | (b & 0x3F))
```

If the first byte of a group matches the bit pattern `1110xxxx`, then the group consists of that byte `a` and two more bytes `b` and `c`. If there is no byte `c` (because byte `a` was one of the last two of the bytes to be read), or either byte `b` or byte `c` does not match the bit pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown. Otherwise, the group is converted to the character:

```
(char) (((a & 0x0F) << 12) | ((b & 0x3F) << 6) | (c & 0x3F))
```

If the first byte of a group matches the pattern `1111xxxx` or the pattern `10xxxxxx`, then a `UTFDataFormatException` is thrown.

If end of file is encountered at any time during this entire process, then an `EOFException` is thrown.

After every group has been converted to a character by this process, the characters are gathered, in the same order in which their corresponding groups were read from the input stream, to form a `String`, which is returned.

The `writeUTF` method of interface `DataOutput` (§22.2.14) may be used to write data that is suitable for reading by this method.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.2 The Interface java.io.DataOutput

The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream. There is also a facility for converting a `String` into Java modified UTF-8 format and writing the resulting series of bytes.

The `DataInput` interface (§22.1) can be used to read in and reconstruct Java data from the binary output data produced by the `DataOutput` interface.

The `DataOutput` interface is implemented by classes `DataOutputStream` (§22.21) and `RandomAccessFile` (§22.23).

```
public interface DataOutput {
    public void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void writeBoolean(boolean v) throws IOException;
    public void writeByte(int v) throws IOException;
    public void writeShort(int v) throws IOException;
    public void writeChar(int v) throws IOException;
    public void writeInt(int v) throws IOException;
    public void writeLong(long v) throws IOException;
    public void writeFloat(float v) throws IOException;
    public void writeDouble(double v) throws IOException;
    public void writeBytes(String s)
        throws IOException, NullPointerException;
    public void writeChars(String s)
        throws IOException, NullPointerException;
    public void writeUTF(String s)
        throws IOException, NullPointerException;
}
```

For all the methods in this interface that write bytes, it is generally true that if a byte cannot be written for any reason, an `IOException` is thrown.

22.2.1 `public void write(int b) throws IOException`

The general contract for `write` is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

22.2.2 `public void write(byte[] b)` `throws IOException, NullPointerException`

The general contract for `write` is that all the bytes in array `b` are written, in order, to the output stream.

If `b` is null, a `NullPointerException` is thrown.

If `b.length` is zero, then no bytes are written. Otherwise, the byte `b[0]` is written first, then `b[1]`, and so on; the last byte written is `b[b.length-1]`.

22.2.3 `public void write(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`, `IndexOutOfBoundsException`

The general contract for `write` is that `len` bytes from array `b` are written, in order, to the output stream.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are written. Otherwise, the byte `b[off]` is written first, then `b[off+1]`, and so on; the last byte written is `b[off+len-1]`.

22.2.4 `public void writeBoolean(boolean v) throws IOException`

The general contract for `writeBoolean` is that one byte is written to the output stream. If the argument `v` is `true`, the value `(byte)1` is written; if `v` is `false`, the value `(byte)0` is written.

The byte written by this method may be read by the `readBoolean` method of interface `DataInput` ([§22.1.4](#)), which will then return a `boolean` equal to `v`.

22.2.5 `public void writeByte(int v) throws IOException`

The general contract for `writeByte` is that one byte is written to the output stream to represent the value of the argument. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored. (This means that `writeByte` does exactly the same thing as `write` for an integer argument.)

The byte written by this method may be read by the `readByte` method of interface `DataInput` ([§22.1.5](#)), which will then return a `byte` equal to `(byte)v`.

22.2.6 `public void writeShort(int v) throws IOException`

The general contract for `writeShort` is that two bytes are written to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte) (0xff & (v >> 8))  
(byte) (0xff & v)
```

The bytes written by this method may be read by the `readShort` method of interface `DataInput` ([§22.1.7](#)), which will then return a `short` equal to `(short)v`.

22.2.7 `public void writeChar(int v) throws IOException`

The general contract for `writeChar` is that two bytes are written to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:


```
(byte) (0xff & (v >> 8))  
(byte) (0xff & v)
```

The bytes written by this method may be read by the `readChar` method of interface `DataInput` ([§22.1.9](#)), which will then return a `char` equal to `(char) v`.

22.2.8 `public void writeInt(int v) throws IOException`

The general contract for `writeInt` is that four bytes are written to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte) (0xff & (v >> 24))  
(byte) (0xff & (v >> 16))  
(byte) (0xff & (v >> 8))  
(byte) (0xff & v)
```

The bytes written by this method may be read by the `readInt` method of interface `DataInput` ([§22.1.10](#)), which will then return an `int` equal to `v`.

22.2.9 `public void writeLong(long v) throws IOException`

The general contract for `writeLong` is that four bytes are written to the output stream to represent the value of the argument. The byte values to be written, in the order shown, are:

```
(byte) (0xff & (v >> 56))  
(byte) (0xff & (v >> 48))  
(byte) (0xff & (v >> 40))  
(byte) (0xff & (v >> 32))  
(byte) (0xff & (v >> 24))  
(byte) (0xff & (v >> 16))  
(byte) (0xff & (v >> 8))  
(byte) (0xff & v)
```

The bytes written by this method may be read by the `readLong` method of interface `DataInput` ([§22.1.11](#)), which will then return a `long` equal to `v`.

22.2.10 `public void writeFloat(float v) throws IOException`

The general contract for `writeFloat` is that four bytes are written to the output stream to represent the value of the argument. It does this as if it first converts this `float` value to an `int` in exactly the manner of the `Float.floatToIntBits` method ([§20.9.22](#)) and then writes the `int` value in exactly the manner of the `writeInt` method ([§22.2.8](#)).

The bytes written by this method may be read by the `readFloat` method of interface `DataInput` ([§22.1.12](#)), which will then return a `float` equal to `v`.

22.2.11 `public void writeDouble(double v) throws IOException`

The general contract for `writeDouble` is that eight bytes are written to the output stream to represent the value of the argument. It does this as if it first converts this double value to a `long` in exactly the manner of the `Double.doubleToLongBits` method (§20.10.21) and then writes the `long` value in exactly the manner of the `writeLong` method (§22.2.9).

The bytes written by this method may be read by the `readDouble` method of interface `DataInput` (§22.1.13), which will then return a `double` equal to `v`.

22.2.12 `public void writeBytes(String s)`
`throws IOException, NullPointerException`

The general contract for `writeBytes` is that for every character in the string `s`, taken in order, one byte is written to the output stream.

If `s` is `null`, a `NullPointerException` is thrown.

If `s.length` is zero, then no bytes are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character written is `s[s.length-1]`. For each character, one byte is written, the low-order byte, in exactly the manner of the `writeByte` method (§22.2.5). The high-order eight bits of each character in the string are ignored.

22.2.13 `public void writeChars(String s)`
`throws IOException, NullPointerException`

The general contract for `writeChars` is that every character in the string `s` is written, in order, to the output stream, two bytes per character.

If `s` is `null`, a `NullPointerException` is thrown.

If `s.length` is zero, then no characters are written. Otherwise, the character `s[0]` is written first, then `s[1]`, and so on; the last character written is `s[s.length-1]`. For each character, two bytes are actually written, high-order byte first, in exactly the manner of the `writeChar` method (§22.2.7).

22.2.14 `public void writeUTF(String s)`
`throws IOException, NullPointerException`

The general contract for `writeUTF` is that two bytes of length information are written to the output stream, followed by the Java modified UTF representation of every character in the string `s`.

If `s` is `null`, a `NullPointerException` is thrown.

Each character in the string `s` is converted to a group of one, two, or three bytes, depending on the value of the character.

If a character `c` is in the range `'\u0001'` through `'\u007f'`, it is represented by one byte:

(byte) `c`

If a character `c` is `'\u0000'` or is in the range `'\u0080'` through `'\u07ff'`, then it is represented by two bytes, to be written in the order shown:


```
(byte) (0xc0 | (0x1f & (c >> 6)))  
(byte) (0x80 | (0x3f & c))
```

If a character `c` is in the range `'\u0800'` through `'\uffff'`, then it is represented by three bytes, to be written in the order shown:

```
(byte) (0xc0 | (0x0f & (c >> 12)))  
(byte) (0x80 | (0x3f & (c >> 6)))  
(byte) (0x80 | (0x3f & c))
```

First, the total number of bytes needed to represent all the characters of `s` is calculated. If this number is larger than 65535, then a `UTFDataFormatError` is thrown. Otherwise, this length is written to the output stream in exactly the manner of the `writeShort` method ([§22.2.6](#)); after this, the one-, two-, or three-byte representation of each character in the string `s` is written.

The bytes written by this method may be read by the `readUTF` method of interface `DataInput` ([§22.1.15](#)), which will then return a `String` equal to `s`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.3 The Class java.io.InputStream

An input stream makes input bytes available from some source.

```
public abstract class InputStream {
    public abstract int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public long skip(long n) throws IOException;
    public int available() throws IOException;
    public void close() throws IOException;
    public void mark(int readlimit);
    public void reset() throws IOException;
    public boolean markSupported();
}
```

22.3.1 `public abstract int read() throws IOException`

The general contract of `read` is that it reads one byte from the input stream. The byte is returned as an integer in the range 0 to 255 (0x00-0xff). If no byte is available because the stream is at end of file, the value -1 is returned.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If the byte cannot be read for any reason other than end of file, an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed ([§22.3.6](#)).

22.3.2 `public int read(byte[] b) throws IOException, NullPointerException`

The general contract of `read(b)` is that it reads some number of bytes from the input stream and stores them into the buffer array `b`. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is null, a `NullPointerException` is thrown.

If the length of `b` is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[0]`, the next one into `b[1]`, and so on. The number of bytes read is, at most, equal to the length of `b`. Let `k` be the number of bytes actually read; these bytes will be stored in elements `b[0]` through `b[k-1]`, leaving elements `b[k]` through `b[b.length-1]` unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed ([§22.15.5](#)).

The `read(b)` method for class `InputStream` has the same effect as:


```
read(b, 0, b.length)
```

22.3.3 `public int read(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`,
 `IndexOutOfBoundsException`

The general contract of `read(b, off, len)` is that it reads some number of bytes from the input stream and stores them into the buffer array `b`. An attempt is made to read as many as `len` bytes, but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If `b` is null, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If `len` is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into `b`.

The first byte read is stored into element `b[off]`, the next one into `b[off+1]`, and so on. The number of bytes read is, at most, equal to `len`. Let k be the number of bytes actually read; these bytes will be stored in elements `b[off]` through `b[off+k-1]`, leaving elements `b[off+k]` through `b[off+len-1]` unaffected.

In every case, elements `b[0]` through `b[off]` and elements `b[off+len]` through `b[b.length-1]` are unaffected.

If the first byte cannot be read for any reason other than end of file, then an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed ([§22.15.5](#)).

The `read(b, off, len)` method for class `InputStream` simply calls the method `read()` repeatedly. If the first such call results in an `IOException`, that exception is returned from the call to the `read(b, off, len)` method. If any subsequent call to `read()` results in a `IOException`, the exception is caught and treated as if it were end of file; the bytes read up to that point are stored into `b` and the number of bytes read before the exception occurred is returned.

22.3.4 `public long skip(long n) throws IOException`

The general contract of `skip` is that it makes an attempt to skip over `n` bytes of data from the input stream, discarding the skipped bytes. However, it may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before `n` bytes have been skipped is only one possibility. The actual number of bytes skipped is returned.

22.3.5 `public int available() throws IOException`

The general contract of `available` is that it returns an integer k ; the next caller of a method for this input stream, which might be the same thread or another thread, can then expect to be able to read or skip up to k bytes without blocking (waiting for input data to arrive).

The `available` method for class `InputStream` always returns 0.

22.3.6 `public int close() throws IOException`

The general contract of `close` is that it closes the input stream. A closed stream cannot perform input operations and cannot be reopened.

The `close` method for class `InputStream` does nothing and simply returns.

22.3.7 `public void mark(int readlimit)`

The general contract of `mark` is that, if the method `markSupported` returns `true`, the stream somehow remembers all the bytes read after the call to `mark` and stands ready to supply those same bytes again if and whenever the method `reset` is called. However, the stream is not required to remember any data at all if more than `readlimit` bytes are read from the stream before `reset` is called.

The `mark` method for class `InputStream` does nothing.

22.3.8 `public void reset() throws IOException`

The general contract of `reset` is:

- If the method `markSupported` returns `true`, then:
 - If the method `mark` has not been called since the stream was created, or the number of bytes read from the stream since `mark` was last called is larger than the argument to `mark` at that last call, then an `IOException` might be thrown.
 - If such an `IOException` is not thrown, then the stream is reset to a state such that all the bytes read since the most recent call to `mark` (or since the start of the file, if `mark` has not been called) will be resupplied to subsequent callers of the `read` method, followed by any bytes that otherwise would have been the next input data as of the time of the call to `reset`.
- If the method `markSupported` returns `false`, then:
 - The call to `reset` may throw an `IOException`.
 - If an `IOException` is not thrown, then the stream is reset to a fixed state that depends on the particular type of the input stream and how it was created. The bytes that will be supplied to subsequent callers of the `read` method depend on the particular type of the input stream.

The method `reset` for class `InputStream` always throws an `IOException`.

22.3.9 `public boolean markSupported()`

The general contract of `markSupported` is that if it returns `true`, then the stream supports the `mark` (§22.3.7) and `reset` (§22.3.8) operations. For any given instance of `InputStream`, this method should consistently return the same truth value whenever it is called.

The `markSupported` method for class `InputStream` returns `false`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.4 The Class java.io.FileInputStream

A file input stream obtains input bytes from a file in a file system. What files are available depends on the host environment.

```
public class FileInputStream extends InputStream {
    public FileInputStream(String path)
        throws SecurityException, FileNotFoundException;
    public FileInputStream(File file)
        throws SecurityException, FileNotFoundException;
    public FileInputStream(FileDescriptor fdObj)
        throws SecurityException;
    public native int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public native long skip(long n) throws IOException;
    public native int available() throws IOException;
    public native void close() throws IOException;
    public final FileDescriptor getFD() throws IOException;
    protected void finalize() throws IOException;
}
```

22.4.1 public FileInputStream(String path)
throws SecurityException, FileNotFoundException

This constructor initializes a newly created `FileInputStream` by opening a connection to an actual file, the file named by the path name `path` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the `path` argument as its argument.

If the actual file cannot be opened, a `FileNotFoundException` is thrown.

22.4.2 public FileInputStream(File file)
throws SecurityException, FileNotFoundException

This constructor initializes a newly created `FileInputStream` by opening a connection to an actual file, the file named by the `File` object `file` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the `path` represented by the `file` argument as its argument.

If the actual file cannot be opened, a `FileNotFoundException` is thrown.

22.4.3 public FileInputStream(FileDescriptor fdObj)
throws SecurityException

This constructor initializes a newly created `FileInputStream` by using the file descriptor `fdObj`, which represents an existing connection to an actual file in the file system.

First, if there is a security manager, its `checkRead` method ([§20.17.18](#)) is called with the file descriptor `fdObj` as its argument.

22.4.4 `public final FileDescriptor getFD() throws IOException`

This method returns the `FileDescriptor` object ([§22.26](#)) that represents the connection to the actual file in the file system being used by this `FileInputStream`.

22.4.5 `public int read() throws IOException;`

The byte for this operation is read from the actual file with which this file input stream is connected.

Implements the `read` method of `InputStream` ([§22.3.1](#)).

22.4.6 `public int read(byte[] b)`
`throws IOException, NullPointerException`

Bytes for this operation are read from the actual file with which this file input stream is connected.

Overrides the `read` method of `InputStream` ([§22.3.2](#)).

22.4.7 `public int read(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

Bytes for this operation are read from the actual file with which this file input stream is connected.

Overrides the `read` method of `InputStream` ([§22.3.3](#)).

22.4.8 `public long skip(long n) throws IOException`

Bytes for this operation are read from the actual file with which this file input stream is connected.

Overrides the `skip` method of `InputStream` ([§22.3.4](#)).

22.4.9 `public int available() throws IOException`

Overrides the `available` method of `InputStream` ([§22.3.5](#)).

22.4.10 `public void close() throws IOException`

This file input stream is closed and may no longer be used for reading bytes.

Overrides the `close` method of `InputStream` ([§22.3.6](#)).

22.4.11 `protected void finalize() throws IOException`

A `FileInputStream` uses finalization to clean up the connection to the actual file.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.5 The Class java.io.PipedInputStream

A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream. Typically, data is read from a `PipedInputStream` object by one thread and data is written to the corresponding `PipedOutputStream` (§22.17) by some other thread. Attempting to use both objects from a single thread is not recommended, as it may deadlock the thread. The piped input stream contains a buffer, decoupling read operations from write operations, within limits.

```
public class PipedInputStream extends InputStream {
    public PipedInputStream (PipedOutputStream src)
        throws IOException;
    public PipedInputStream ();
    public void connect(PipedOutputStream src)
        throws IOException;
    public int read() throws IOException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void close() throws IOException;
}
```

22.5.1 `public PipedInputStream(PipedOutputStream src)`
throws `IOException`

This constructor initializes a newly created `PipedInputStream` so that it is connected to the piped output stream `src`. Data bytes written to `src` will then be available as input from this stream.

22.5.2 `public PipedInputStream()`

This constructor initializes a newly created `PipedInputStream` so that it is not yet connected. It must be connected to a `PipedOutputStream` before being used.

22.5.3 `public void connect(PipedOutputStream src)`
throws `IOException`

The `connect` method causes this piped input stream to be connected to the piped output stream `src`. If this object is already connected to some other piped output stream, an `IOException` is thrown.

If `src` is an unconnected piped output stream and `snk` is an unconnected piped input stream, they may be connected by either the call:

```
snk.connect(src)
```

or the call:

```
src.connect(snk)
```


The two calls have the same effect.

22.5.4 `public int read() throws IOException`

If a thread was providing data bytes to the connected piped output stream, but the thread is no longer alive, then an `IOException` is thrown.

Implements the `read` method of `InputStream` ([§22.3.1](#)).

22.5.5 `public int read(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

If a thread was providing data bytes to the connected piped output stream, but the thread is no longer alive, then an `IOException` is thrown.

Overrides the `read` method of `InputStream` ([§22.3.3](#)).

22.5.6 `public void close() throws IOException`

This piped input stream is closed and may no longer be used for reading bytes.

Overrides the `close` method of `InputStream` ([§22.3.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.6 The Class java.io.ByteArrayInputStream

A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the `read` method. See also `StringBufferInputStream` ([§22.7](#)).

```
public class ByteArrayInputStream extends InputStream {
    protected byte[] buf;
    protected int pos;
    protected int count;
    public ByteArrayInputStream(byte[] buf);
    public ByteArrayInputStream(byte[] buf,
                                int offset, int length);
    public int read()
        throws NullPointerException, IndexOutOfBoundsException;
    public int read(byte[] b, int off, int len)
        throws NullPointerException, IndexOutOfBoundsException;
    public long skip(long n);
    public int available();
    public void reset();
}
```

22.6.1 `protected byte[] buf;`

An array of bytes that was provided by the creator of the stream. Elements `buf[0]` through `buf[count-1]` are the only bytes that can ever be read from the stream; element `buf[pos]` is the next byte to be read.

22.6.2 `protected int pos;`

This value should always be nonnegative and not larger than the value of `count`. The next byte to be read from this stream will be `buf[pos]`.

22.6.3 `protected int count;`

This value should always be nonnegative and not larger than the length of `buf`. It is one greater than the position of the last byte within `buf` that can ever be read from this stream.

22.6.4 `public ByteArrayInputStream(byte[] buf)`

This constructor initializes a newly created `ByteArrayInputStream` so that it uses `buf` as its buffer array. The initial value of `pos` is 0 and the initial value of `count` is the length of `buf`.

22.6.5 `public ByteArrayInputStream(byte[] buf,
int offset, int length)`

This constructor initializes a newly created `ByteArrayInputStream` so that it uses `buf` as its buffer array. The initial value of `pos` is `offset` and the initial value of `count` is `offset+len`.

Note that if bytes are simply read from the resulting input stream, elements `buf[pos]` through `buf[pos+len-1]` will be read; however, if a `reset` operation (§22.6.10) is performed, then bytes `buf[0]` through `buf[pos-1]` will then become available for input.

22.6.6 `public int read()`
throws `NullPointerException`, `IndexOutOfBoundsException`

If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the value `buf[pos]&0xff` is returned; just before the return, `pos` is incremented by 1.

Implements the `read` method of `InputStream` (§22.3.1).

22.6.7 `public int read(byte[] b, int off, int len)`
throws `NullPointerException`, `IndexOutOfBoundsException`

If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the number `k` of bytes read is equal to the smaller of `len` and `count-pos`. If `k` is positive, then bytes `buf[pos]` through `buf[pos+k-1]` are copied into `b[off]` through `b[off+k-1]` in the manner performed by `System.arraycopy` (§20.18.16). The value `k` is added into `pos` and `k` is returned.

Overrides the `read` method of `InputStream` (§22.3.3).

22.6.8 `public long skip(long n)`

The actual number `k` of bytes to be skipped is equal to the smaller of `n` and `count-pos`. The value `k` is added into `pos` and `k` is returned.

Overrides the `skip` method of `InputStream` (§22.3.4).

22.6.9 `public int available()`

The quantity `count-pos` is returned.

Overrides the `available` method of `InputStream` (§22.3.5).

22.6.10 `public void reset()`

The value of `pos` is set to 0.

Overrides the `reset` method of `InputStream` (§22.3.8).

{`ewl msdncd.dll, ewcright, /c"Microsoft"`}

22.7 The Class `java.io.StringBufferInputStream`

A `StringBufferInputStream` contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the `read` method. See also `ByteArrayInputStream` ([§22.6](#)).

```
public class StringBufferInputStream extends InputStream {
    protected String buffer;
    protected int pos;
    protected int count;
    public StringBufferInputStream(String s)
        throws NullPointerException;
    public int read();
    public int read(byte[] b, int off, int len)
        throws NullPointerException, IndexOutOfBoundsException;
    public long skip(long n);
    public int available();
    public void reset();
}
```

Note that bytes read from a `StringBufferInputStream` are the low-order eight bits of each character in the string; the high-order eight bits of each character are ignored.

22.7.1 `protected String buffer;`

A `String` that was provided by the creator of the stream. Elements `buffer[0]` through `buffer[count-1]` are the only bytes that can ever be read from this stream; element `buffer[pos]` is the next byte to be read.

22.7.2 `protected int pos;`

This value should always be nonnegative and not larger than the value of `count`. The next byte to be read from this stream will be `buffer[pos]`.

22.7.3 `protected int count;`

This value equals the length of `buffer`. It is the number of bytes of data in `buffer` that can ever be read from this stream.

22.7.4 `public StringBufferInputStream(String s)`
`throws NullPointerException`

This constructor initializes a newly created `StringBufferInputStream` so that it uses `s` as its buffer array. The initial value of `pos` is 0 and the initial value of `count` is the length of `buffer`.

22.7.5 `public int read()`

If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the value `buffer[pos]&0xff` is returned; just before the return, 1 is added to `pos`.

Implements the `read` method of `InputStream` ([§22.3.1](#)).

22.7.6 `public int read(byte[] b, int off, int len)`
throws `NullPointerException`, `IndexOutOfBoundsException`

If `pos` equals `count`, then `-1` is returned to indicate end of file. Otherwise, the number `k` of bytes read is equal to the smaller of `len` and `count-pos`. If `k` is positive, then bytes `buffer[pos]` through `buffer[pos+k-1]` are copied into `b[off]` through `b[off+k-1]` in the manner performed by `System.arraycopy` ([§20.18.16](#)). The value `k` is added into `pos` and `k` is returned.

Overrides the `read` method of `InputStream` ([§22.3.3](#)).

22.7.7 `public long skip(long n)`

The actual number `k` of bytes to be skipped is equal to the smaller of `n` and `count-pos`. The value `k` is added into `pos` and `k` is returned.

Overrides the `skip` method of `InputStream` ([§22.3.4](#)).

22.7.8 `public int available()`

The quantity `count-pos` is returned.

Overrides the `available` method of `InputStream` ([§22.3.5](#)).

22.7.9 `public void reset()`

The value of `pos` is set to `0`.

Overrides the `reset` method of `InputStream` ([§22.3.8](#)).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

22.8 The Class java.io.SequenceInputStream

A `SequenceInputStream` represents the logical concatenation of other input streams. It starts out with an ordered collection of input streams and reads from the first one until end of file is reached, whereupon it reads from the second one, and so on, until end of file is reached on the last of the contained input streams.

```
public class SequenceInputStream extends InputStream {
    public SequenceInputStream(Enumeration e);
    public SequenceInputStream(InputStream s1, InputStream s2);
    public int read() throws IOException;
    public int read(byte[] buf, int pos, int len)
        throws IOException, NullPointerException,
        IndexOutOfBoundsException;
    public void close() throws IOException;
}
```

22.8.1 `public SequenceInputStream(Enumeration e)`

This constructor initializes a newly created `SequenceInputStream` by remembering the argument, which must be an `Enumeration` (§21.1) that produces objects whose run-time type is `InputStream` (§22.3). The input streams that are produced by the enumeration will be read, in order, to provide the bytes to be read from this `SequenceInputStream`. After each input stream from the enumeration is exhausted, it is closed by calling its `close` method.

22.8.2 `public SequenceInputStream(InputStream s1, InputStream s2)`

This constructor initializes a newly created `SequenceInputStream` by remembering the two arguments, which will be read in order, first `s1` and then `s2`, to provide the bytes to be read from this `SequenceInputStream`.

22.8.3 `public int read() throws IOException`

Implements the `read` method of `InputStream` (§22.3.1).

22.8.4 `public int read(byte[] buf, int pos, int len) throws IOException, NullPointerException, IndexOutOfBoundsException`

Overrides the `read` method of `InputStream` (§22.3.3).

22.8.5 `public void close() throws IOException`

This `SequenceInputStream` is closed. A closed `SequenceInputStream` cannot perform input operations and cannot be reopened.

If this stream was created from an enumeration, all remaining elements are requested from the enumeration and closed before the `close` method returns.

Overrides the `close` method of `InputStream` ([§22.3.6](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.9 The Class java.io.FilterInputStream

A `FilterInputStream` contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality. The class `FilterInputStream` itself simply overrides all methods of `InputStream` with versions that pass all requests to the contained input stream. Subclasses of `FilterInputStream` may further override some of these methods and may also provide additional methods and fields.

```
public class FilterInputStream extends InputStream {
    protected InputStream in;
    protected FilterInputStream(InputStream in);
    public int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public long skip(long n) throws IOException;
    public int available() throws IOException;
    public void close() throws IOException;
    public void mark(int readlimit);
    public void reset() throws IOException;
    public boolean markSupported();
}
```

22.9.1 `protected InputStream in;`

The input stream to be filtered.

22.9.2 `protected FilterInputStream(InputStream in)`

This constructor initializes a newly created `FilterInputStream` by assigning the argument `in` to the field `this.in` so as to remember it for later use.

22.9.3 `public int read() throws IOException`

This method simply performs `in.read()` and returns the result.

Implements the `read` method of `InputStream` ([§22.3.1](#)).

22.9.4 `public int read(byte[] b)`
`throws IOException, NullPointerException`

This method simply performs the call `read(b, 0, b.length)` and returns the result. It is important that it does *not* do `in.read(b)` instead; certain subclasses of `FilterInputStream` depend on the implementation strategy actually used.

Overrides the `read` method of `InputStream` ([§22.3.2](#)).

22.9.5 `public int read(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`, `IndexOutOfBoundsException`

This method simply performs `in.read(b, off, len)` and returns the result.

Overrides the `read` method of `InputStream` ([§22.3.3](#)).

22.9.6 `public long skip(long n)` throws `IOException`

This method simply performs `in.skip()` and returns the result.

Overrides the `skip` method of `InputStream` ([§22.3.4](#)).

22.9.7 `public int available()` throws `IOException`

This method simply performs `in.available()` and returns the result.

Overrides the `available` method of `InputStream` ([§22.3.5](#)).

22.9.8 `public void close()` throws `IOException`

This method simply performs `in.close()`.

Overrides the `close` method of `InputStream` ([§22.3.6](#)).

22.9.9 `public void mark(int readlimit)`

This method simply performs `in.mark()`.

Overrides the `mark` method of `InputStream` ([§22.3.7](#)).

22.9.10 `public void reset()` throws `IOException`

This method simply performs `in.reset()`.

Overrides the `reset` method of `InputStream` ([§22.3.8](#)).

22.9.11 `public boolean markSupported()`

This method simply performs `in.markSupported()` and returns whatever value is returned from that invocation.

Overrides the `markSupported` method of `InputStream` ([§22.3.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.10 The Class java.io.BufferedInputStream

A `BufferedInputStream` adds functionality to another input stream—namely, the ability to buffer the input and to support the `mark` and `reset` methods. When the `BufferedInputStream` is created, an internal buffer array is created. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time. The `mark` operation remembers a point in the input stream and the `reset` operation causes all the bytes read since the most recent `mark` operation to be reread before new bytes are taken from the contained input stream.

```
public class BufferedInputStream extends FilterInputStream {
    protected byte[] buf;
    protected int count = 0;
    protected int pos = 0;
    protected int markpos = -1;
    protected int marklimit = 0;
    public BufferedInputStream(InputStream in);
    public BufferedInputStream(InputStream in, int size);
    public int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public long skip(long n) throws IOException;
    public int available() throws IOException;
    public void mark(int readlimit);
    public void reset() throws IOException;
    public boolean markSupported();
}
```

22.10.1 `protected byte[] buf;`

The internal buffer array. When necessary, it may be replaced by another array of a different size.

22.10.2 `protected int count = 0;`

This value is always in the range 0 through `buf.length`; elements `buf[0]` through `buf[count-1]` contain buffered input data obtained from the underlying input stream.

22.10.3 `protected int pos = 0;`

This value is always in the range 0 through `count`. If it is less than `count`, then `buf[pos]` is the next byte to be supplied as input; if it is equal to `count`, then the next `read` or `skip` operation will require more bytes to be read from the contained input stream.

22.10.4 `protected int markpos = -1;`

This value is always in the range -1 through `pos`. If there is no marked position in the input stream,

this field is -1. If there is a marked position in the input stream, then `buf[markpos]` is the first byte to be supplied as input after a `reset` operation. If `markpos` is not -1, then all bytes from positions `buf[markpos]` through `buf[pos-1]` must remain in the buffer array (though they may be moved to another place in the buffer array, with suitable adjustments to the values of `count`, `pos`, and `markpos`); they may not be discarded unless and until the difference between `pos` and `markpos` exceeds `marklimit`.

22.10.5 `protected int marklimit;`

Whenever the difference between `pos` and `markpos` exceeds `marklimit`, then the mark may be dropped by setting `markpos` to -1.

22.10.6 `public BufferedInputStream(InputStream in)`

This constructor initializes a newly created `BufferedInputStream` by saving its argument, the input stream `in`, for later use. An internal buffer array is created and stored in `buf`.

22.10.7 `public BufferedInputStream(InputStream in, int size)`

This constructor initializes a newly created `BufferedInputStream` by saving its argument, the input stream `in`, for later use. An internal buffer array of length `size` is created and stored in `buf`.

22.10.8 `public int read() throws IOException`

See the general contract of the `read` method of `InputStream` ([§22.3.1](#)).

Overrides the `read` method of `FilterInputStream` ([§22.9.3](#)).

22.10.9 `public int read(byte[] b)`
`throws IOException, NullPointerException`

See the general contract of the `read` method of `InputStream` ([§22.3.2](#)).

Overrides the `read` method of `FilterInputStream` ([§22.9.4](#)).

22.10.10 `public int read(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `read` method of `InputStream` ([§22.3.3](#)).

Overrides the `read` method of `FilterInputStream` ([§22.9.5](#)).

22.10.11 `public long skip(long n) throws IOException`

See the general contract of the `skip` method of `InputStream` ([§22.3.4](#)).

Overrides the `skip` method of `FilterInputStream` ([§22.9.6](#)).

22.10.12 `public int available() throws IOException`

See the general contract of the `available` method of `InputStream` ([§22.3.5](#)).

Overrides the `available` method of `FilterInputStream` ([§22.9.7](#)).

22.10.13 `public void mark(int readlimit)`

The field `marklimit` is set equal to the argument and `markpos` is set equal to `pos`

Overrides the `mark` method of `FilterInputStream` ([§22.9.9](#)).

22.10.14 `public void reset() throws IOException`

See the general contract of the `reset` method of `InputStream` ([§22.3.8](#)).

If `markpos` is `-1` (no mark has been set or the mark has been invalidated), an `IOException` is thrown. Otherwise, `pos` is set equal to `markpos`.

Overrides the `reset` method of `FilterInputStream` ([§22.9.10](#)).

22.10.15 `public boolean markSupported()`

This method returns `true` (a `BufferedInputStream` always supports `mark`).

Overrides the `markSupported` method of `FilterInputStream` ([§22.9.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.11 The Class java.io.DataInputStream

A data input stream provides facilities for reading bytes from an input source and interpreting specific character sequences as representing data of diverse types.

```
public class DataInputStream extends FilterInputStream
    implements DataInput {
    public DataInputStream(InputStream in);
    public final void readFully(byte[] b)
        throws IOException, NullPointerException;
    public final void readFully(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public final int skipBytes(int n) throws IOException;
    public final boolean readBoolean() throws IOException;
    public final byte readByte() throws IOException;
    public final int readUnsignedByte() throws IOException;
    public final short readShort() throws IOException;
    public final int readUnsignedShort() throws IOException;
    public final char readChar() throws IOException;
    public final int readInt() throws IOException;
    public final long readLong() throws IOException;
    public final float readFloat() throws IOException;
    public final double readDouble() throws IOException;
    public final String readLine() throws IOException;
    public final String readUTF() throws IOException;
    public final static String readUTF(DataInput in)
        throws IOException;
}
```

22.11.1 `public DataInputStream(InputStream in)`

This constructor initializes a newly created `DataInputStream` by saving its argument, the input stream `in`, for later use.

22.11.2 `public final void readFully(byte[] b)`
`throws IOException, NullPointerException`

See the general contract of the `readFully` method of `DataInput` ([§22.1.1](#)).

Bytes for this operation are read from the contained input stream.

22.11.3 `public final void readFully(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `readFully` method of `DataInput` ([§22.1.2](#)).

Bytes for this operation are read from the contained input stream.

22.11.4 `public final int skipBytes(int n) throws IOException`

See the general contract of the `skipBytes` method of `DataInput` ([§22.1.3](#)).

Bytes for this operation are read from the contained input stream.

22.11.5 `public final boolean readBoolean() throws IOException`

See the general contract of the `readBoolean` method of `DataInput` ([§22.1.4](#)).

The byte for this operation is read from the contained input stream.

22.11.6 `public final byte readByte() throws IOException`

See the general contract of the `readByte` method of `DataInput` ([§22.1.5](#)).

The byte for this operation is read from the contained input stream.

22.11.7 `public final int readUnsignedByte() throws IOException`

See the general contract of the `readUnsignedByte` method of `DataInput` ([§22.1.6](#)).

The byte for this operation is read from the contained input stream.

22.11.8 `public final short readShort() throws IOException`

See the general contract of the `readShort` method of `DataInput` ([§22.1.7](#)).

Bytes for this operation are read from the contained input stream.

22.11.9 `public final int readUnsignedShort() throws IOException`

See the general contract of the `readUnsignedShort` method of `DataInput` ([§22.1.8](#)).

Bytes for this operation are read from the contained input stream.

22.11.10 `public final char readChar() throws IOException`

See the general contract of the `readChar` method of `DataInput` ([§22.1.9](#)).

Bytes for this operation are read from the contained input stream.

22.11.11 `public final int readInt() throws IOException`

See the general contract of the `readInt` method of `DataInput` ([§22.1.10](#)).

Bytes for this operation are read from the contained input stream.

22.11.12 `public final long readLong() throws IOException`

See the general contract of the `readLong` method of `DataInput` ([§22.1.11](#)).

Bytes for this operation are read from the contained input stream.

22.11.13 `public final float readFloat() throws IOException`

See the general contract of the `readFloat` method of `DataInput` ([§22.1.12](#)).

Bytes for this operation are read from the contained input stream.

22.11.14 `public final double readDouble() throws IOException`

See the general contract of the `readDouble` method of `DataInput` ([§22.1.13](#)).

Bytes for this operation are read from the contained input stream.

22.11.15 `public final String readLine() throws IOException`

See the general contract of the `readLine` method of `DataInput` ([§22.1.14](#)).

Bytes for this operation are read from the contained input stream.

22.11.16 `public final String readUTF() throws IOException`

See the general contract of the `readUTF` method of `DataInput` ([§22.1.15](#)).

Bytes for this operation are read from the contained input stream.

22.11.17 `public final static String readUTF(DataInput in)`
`throws IOException`

The `readUTF` method reads from the stream `in` a representation of a Unicode character string encoded in Java modified UTF-8 format; this string of characters is then returned as a `String`. The details of the modified UTF-8 representation are exactly the same as for the `readUTF` method of `DataInput` ([§22.1.15](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.12 The Class java.io.LineNumberInputStream

A `LineNumberInputStream` adds functionality to another input stream, namely the ability to count lines. When the `LineNumberInputStream` is created, the line number counter is set to zero. As bytes from the stream are read or skipped, the counter is incremented whenever a line terminator (`\n`, `\r`, or `\r\n`) is encountered. Such line terminators are also converted to a single `'\n'` character. The method `getLineNumber` returns the current value of the counter, and the method `setLineNumber` sets the counter to a given integer value. If the contained input stream supports the `mark` operation, then so does the `LineNumberInputStream`; the `mark` operation remembers the line number counter and the `reset` operation sets the counter to the value remembered by the `mark` operation.

```
public class LineNumberInputStream extends FilterInputStream {
    public LineNumberInputStream(InputStream in);
    public int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public long skip(long n) throws IOException;
    public int available() throws IOException;
    public void mark(int readlimit);
    public void reset() throws IOException;
    public int getLineNumber();
    public void setLineNumber(int lineNumber);
}
```

22.12.1 `public LineNumberInputStream(InputStream in)`

This constructor initializes a newly created `LineNumberInputStream` by saving its argument, the input stream `in`, for later use.

22.12.2 `public int read() throws IOException`

See the general contract of the `read` method of `InputStream` ([§22.3.1](#)).

As bytes are read from the contained input stream, line terminators are recognized and counted. For each line terminator recognized in the contained input stream, a single character `'\n'` is returned.

Overrides the `read` method of `FilterInputStream` ([§22.9.3](#)).

22.12.3 `public int read(byte[] b)`
`throws IOException, NullPointerException`

See the general contract of the `read` method of `InputStream` ([§22.3.2](#)).

As bytes are read from the contained input stream, line terminators are recognized and counted. For each line terminator recognized in the contained input stream, a single character `'\n'` is returned.

Overrides the `read` method of `FilterInputStream` ([§22.9.4](#)).

22.12.4 `public int read(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`, `IndexOutOfBoundsException`

See the general contract of the `read` method of `InputStream` ([§22.3.3](#)).

As bytes are read from the contained input stream, line terminators are recognized and counted. For each line terminator recognized in the contained input stream, a single character `'\n'` is returned.

Overrides the `read` method of `FilterInputStream` ([§22.9.5](#)).

22.12.5 `public long skip(long n)` throws `IOException`

See the general contract of the `skip` method of `InputStream` ([§22.3.4](#)).

As bytes are read from the contained input stream, line terminators are recognized and counted. Each line terminator recognized in the contained input stream is considered to be a single byte skipped, even if it is the sequence `\r\n`.

Overrides the `skip` method of `FilterInputStream` ([§22.9.6](#)).

22.12.6 `public int available()` throws `IOException`

See the general contract of the `available` method of `InputStream` ([§22.3.5](#)).

Note that if the contained input stream is able to supply k input characters without blocking, the `LineNumberInputStream` can guarantee only to provide `{ewc msdn cd, EWGraphic, LNG12z 0 /a "langref.BMP"}` characters without blocking, because the k characters from the contained input stream might consist of `{ewc msdn cd, EWGraphic, LNG12z 1 /a "langref.BMP"} \r\n` pairs, which will be converted to just `{ewc msdn cd, EWGraphic, LNG12z 2 /a "langref.BMP"} '\n'` characters.

Overrides the `available` method of `FilterInputStream` ([§22.9.7](#)).

22.12.7 `public void mark(int readlimit)`

See the general contract of the `mark` method of `InputStream` ([§22.3.7](#)).

Marking a point in the input stream remembers the current line number as it would be returned by `getLineNumber` ([§22.12.9](#)).

Overrides the `mark` method of `FilterInputStream` ([§22.9.9](#)).

22.12.8 `public void reset()` throws `IOException`

See the general contract of the `reset` method of `InputStream` ([§22.3.8](#)).

Resetting the input stream to a previous point also resets the line number to the value it had at the marked point.

Overrides the `reset` method of `FilterInputStream` ([§22.9.10](#)).

22.12.9 `public int getLineNumber()`

The current line number is returned. This quantity depends on k , the number of line terminators encountered since the most recent occurrence of one of the following three kinds of events:

- If a call to the `setLineNumber` method was most recent, let n be the argument that was given to `setLineNumber`; then the current line number is {ewc msdncd, EWGraphic, LNG12z 3 /a "langref.BMP"}.
- If a call to the `reset` method was most recent, let m be the line number that had been remembered by `mark`; then the current line number is {ewc msdncd, EWGraphic, LNG12z 4 /a "langref.BMP"}.
- If creation of the `LineNumberInputStream` was most recent (that is, neither of the other kinds of event have occurred), then the current line number is k .

These rules imply that the current line number is 0 as the characters of the first line are read, and becomes 1 after the line terminator for the first line has been read.

22.12.10 `public void setLineNumber(int lineNumber)`

The current line number is set equal to the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.13 The Class java.io.PushbackInputStream

A `PushbackInputStream` adds functionality to another input stream, namely the ability to "push back" or "unread" one byte. This is useful in situations where it is convenient for a fragment of code to read an indefinite number of data bytes that are delimited by a particular byte value; after reading the terminating byte, the code fragment can "unread" it, so that the next read operation on the input stream will reread the byte that was pushed back. For example, bytes representing the characters constituting an identifier might be terminated by a byte representing an operator character; a method whose job is to read just an identifier can read until it sees the operator and then push the operator back to be re-read.

```
public class PushbackInputStream extends FilterInputStream {
    protected int    pushBack = -1;
    public PushbackInputStream(InputStream in);
    public int read() throws IOException;
    public int read(byte[] bytes, int offset, int length)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void unread(int ch) throws IOException;
    public int available() throws IOException;
    public boolean markSupported();
}
```

22.13.1 `protected int pushBack = -1;`

If this field has a nonnegative value, it is a byte that was pushed back. If this field is `-1`, there is currently no pushed-back byte.

22.13.2 `public PushbackInputStream(InputStream in)`

This constructor initializes a newly created `PushbackInputStream` by saving its argument, the input stream `in`, for later use. Initially, there is no pushed-back byte (the field `pushBack` is initialized to `-1`).

22.13.3 `public int read() throws IOException`

See the general contract of the `read` method of `InputStream` ([§22.3.1](#)).

If `pushBack` is not `-1`, the value of `pushBack` is returned and `pushBack` is set to `-1`. Otherwise, a byte is obtained from the contained input stream.

Overrides the `read` method of `FilterInputStream` ([§22.9.3](#)).

22.13.4 `public int read(byte[] bytes, int offset, int length) throws
IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `read` method of `InputStream` ([§22.3.3](#)).

If `pushBack` is not `-1`, it is used as an input byte (and `pushBack` is set to `-1`) before any bytes are read from the contained input stream.

Overrides the `read` method of `FilterInputStream` ([§22.9.5](#)).

22.13.5 `public void unread(int b) throws IOException`

If `pushBack` is not `-1`, an `IOException` is thrown (it is not permitted to push back more than one byte). Otherwise, the byte value `b` is pushed back by assigning `b` to `pushBack`.

22.13.6 `public int available() throws IOException`

See the general contract of the `available` method of `InputStream` ([§22.3.1](#)).

This method first calls the `available` method of the contained input stream. If `pushBack` is `-1`, the result is returned; otherwise, the result plus 1 is returned.

Overrides the `available` method of `FilterInputStream` ([§22.9.7](#)).

22.13.7 `public boolean markSupported()`

This method returns `false` (a `PushbackInputStream` does not support `mark`).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

22.14 The Class java.io.StreamTokenizer

A `StreamTokenizer` takes an input stream and parses it into "tokens," allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states, allowing recognition of identifiers, numbers, quoted strings, and comments in a standard style.

```
public class StreamTokenizer {
    public static final int TT_EOF = -1;
    public static final int TT_EOL = '\n';
    public static final int TT_NUMBER = -2;
    public static final int TT_WORD = -3;
    public int ttype;
    public String sval;
    public double nval;
    public StreamTokenizer (InputStream in);
    public void resetSyntax();
    public void wordChars(int low, int hi);
    public void whitespaceChars(int low, int hi);
    public void ordinaryChars(int low, int hi);
    public void ordinaryChar(int ch);
    public void commentChar(int ch);
    public void quoteChar(int ch);
    public void parseNumbers();
    public void eolIsSignificant(boolean flag);
    public void slashStarComments(boolean flag);
    public void slashSlashComments(boolean flag);
    public void lowerCaseMode(boolean flag);
    public int nextToken() throws IOException;
    public void pushBack();
    public int lineno();
    public String toString();
}
```

Each byte read from the input stream is regarded as a character in the range '`\u0000`' through '`\u00FF`'. The character value is used to look up five possible attributes of the character: whitespace, alphabetic, numeric, string quote, and comment character (a character may have more than one of these attributes, or none at all). In addition, there are three flags controlling whether line terminators are to be recognized as tokens, whether Java-style end-of-line comments that start with `//` should be recognized and skipped, and whether Java-style "traditional" comments delimited by `/*` and `*/` should be recognized and skipped. One more flag controls whether all the characters of identifiers are converted to lowercase.

Here is a simple example of the use of a `StreamTokenizer`. The following code merely reads all the tokens in the standard input stream and prints an identification of each one. Changes in the line number are also noted.

```
import java.io.StreamTokenizer;
import java.io.IOException;
```



```

class Tok {
    public static void main(String[] args) {
        StreamTokenizer st = new StreamTokenizer(System.in);
        st.ordinaryChar('/');
        int lineNum = -1;
        try {
            for (int tokenType = st.nextToken();
                tokenType != StreamTokenizer.TT_EOF;
                tokenType = st.nextToken()) {
                int newLineNum = st.lineno();
                if (newLineNum != lineNum) {
                    System.out.println("[line " + newLineNum
                                         + "]\n");
                    lineNum = newLineNum;
                }
                switch(tokenType) {
                    case StreamTokenizer.TT_NUMBER:
                        System.out.println("the number " + st.nval);
                        break;
                    case StreamTokenizer.TT_WORD:
                        System.out.println("identifier " + st.sval);
                        break;
                    default:
                        System.out.println("  operator " +
                                           (char)tokenType);
                }
            }
        } catch (IOException e) {
            System.out.println("I/O failure");
        }
    }
}

```

If the input stream contains this data:

```

10 LET A = 4.5
20 LET B = A*A
30 PRINT A, B

```

then the resulting output is:

```

[line 1]
the number 10.0
identifier LET
identifier A
  operator =
the number 4.5
[line 2]
the number 20.0

```



```
identifier LET
identifier B
operator =
identifier A
operator *
identifier A
[line 3]
the number 30.0
identifier PRINT
identifier A
operator ,
identifier B
```

22.14.1 `public static final int TT_EOF = -1;`

A constant that indicates end of file was reached.

22.14.2 `public static final int TT_EOL = '\n';`

A constant that indicates that a line terminator was recognized.

22.14.3 `public static final int TT_NUMBER = -2;`

A constant that indicates that a number was recognized.

22.14.4 `public static final int TT_WORD = -3;`

A constant that indicates that a word (identifier) was recognized.

22.14.5 `public int ttype;`

The type of the token that was last recognized by this `StreamTokenizer`. This will be `TT_EOF`, `TT_EOL`, `TT_NUMBER`, `TT_WORD`, or a nonnegative byte value that was the first byte of the token (for example, if the token is a string token, then `ttype` has the quote character that started the string).

22.14.6 `public String sval;`

If the value of `ttype` is `TT_WORD` or a string quote character, then the value of `sval` is a `String` that contains the characters of the identifier or of the string (without the delimiting string quotes). For all other types of tokens recognized, the value of `sval` is `null`.

22.14.7 `public double nval;`

If the value of `ttype` is `TT_NUMBER`, then the value of `nval` is the numerical value of the number.

22.14.8 `public StreamTokenizer(InputStream in)`

This constructor initializes a newly created `StreamTokenizer` by saving its argument, the input stream `in`, for later use. The `StreamTokenizer` is also initialized to the following default state:

- All byte values 'A' through 'Z', 'a' through 'z', and 0xA0 through 0xFF are considered to be alphabetic.
- All byte values 0x00 through 0x20 are considered to be whitespace.
- '/' is a comment character.
- Single quote '\'' and double quote '"' are string quote characters.
- Numbers are parsed.
- End of line is not significant.
- // comments and /* comments are not recognized.

22.14.9 `public void resetSyntax()`

The syntax table for this `StreamTokenizer` is reset so that every byte value is "ordinary"; thus, no character is recognized as being a whitespace, alphabetic, numeric, string quote, or comment character. Calling this method is therefore equivalent to:

```
ordinaryChars(0x00, 0xff)
```

The three flags controlling recognition of line terminators, // comments, and /* comments are unaffected.

22.14.10 `public void wordChars(int low, int hi)`

The syntax table for this `StreamTokenizer` is modified so that every character in the range `low` through `hi` has the "alphabetic" attribute.

22.14.11 `public void whitespaceChars(int low, int hi)`

The syntax table for this `StreamTokenizer` is modified so that every character in the range `low` through `hi` has the "whitespace" attribute.

22.14.12 `public void ordinaryChars(int low, int hi)`

The syntax table for this `StreamTokenizer` is modified so that every character in the range `low` through `hi` has no attributes.

22.14.13 `public void ordinaryChar(int ch)`

The syntax table for this `StreamTokenizer` is modified so that the character `ch` has no attributes.

22.14.14 `public void commentChar(int ch)`

The syntax table for this `StreamTokenizer` is modified so that the character `ch` has the "comment character" attribute.

22.14.15 `public void quoteChar(int ch)`

The syntax table for this `StreamTokenizer` is modified so that the character `ch` has the "string

quote" attribute.

22.14.16 `public void parseNumbers()`

The syntax table for this `StreamTokenizer` is modified so that each of the twelve characters

0 1 2 3 4 5 6 7 8 9 . -

has the "numeric" attribute.

22.14.17 `public void eolIsSignificant(boolean flag)`

This `StreamTokenizer` henceforth recognizes line terminators as tokens if and only if the `flag` argument is `true`.

22.14.18 `public void slashStarComments(boolean flag)`

This `StreamTokenizer` henceforth recognizes and skips Java-style "traditional" comments, which are delimited by `/*` and `*/` and do not nest, if and only if the `flag` argument is `true`.

22.14.19 `public void slashSlashComments(boolean flag)`

This `StreamTokenizer` henceforth recognizes and skips Java-style end-of-line comments that start with `//` if and only if the `flag` argument is `true`.

22.14.20 `public void lowerCaseMode(boolean flag)`

This `StreamTokenizer` henceforth converts all the characters in identifiers to lowercase if and only if the `flag` argument is `true`.

22.14.21 `public int nextToken() throws IOException`

If the previous token was pushed back ([§22.14.22](#)), then the value of `tttype` is returned, effectively causing that same token to be reread.

Otherwise, this method parses the next token in the contained input stream. The type of the token is returned; this same value is also made available in the `tttype` field, and related data may be made available in the `sval` and `nval` fields.

First, whitespace characters are skipped, except that if a line terminator is encountered and this `StreamTokenizer` is currently recognizing line terminators, then the type of the token is `TT_EOL`.

If a numeric character is encountered, then an attempt is made to recognize a number. If the first character is `'-'` and the next character is not numeric, then the `'-'` is considered to be an ordinary character and is recognized as a token in its own right. Otherwise, a number is parsed, stopping before the next occurrence of `'-'`, the second occurrence of `'.'`, the first nonnumeric character encountered, or end of file, whichever comes first. The type of the token is `TT_NUMBER` and its value is made available in the field `nval`.

If an alphabetic character is encountered, then an identifier is recognized, consisting of that character and all following characters up to, but not including, the first character that is neither alphabetic nor

numeric, or up to end of file, whichever comes first. The characters of the identifier may be converted to lowercase if this `StreamTokenizer` is in lowercase mode.

If a comment character is encountered, then all subsequent characters are skipped and ignored, up to but not including the next line terminator or end of file. Then another attempt is made to recognize a token. If this `StreamTokenizer` is currently recognizing line terminators, then a line terminator that ends a comment will be recognized as a token in the same manner as any other line terminator in the contained input stream.

If a string quote character is encountered, then a string is recognized, consisting of all characters after (but not including) the string quote character, up to (but not including) the next occurrence of that same string quote character, or a line terminator, or end of file. The usual escape sequences (§3.10.6) such as `\n` and `\t` are recognized and converted to single characters as the string is parsed.

If `//` is encountered and this `StreamTokenizer` is currently recognizing `//` comments, then all subsequent characters are skipped and ignored, up to but not including the next line terminator or end of file. Then another attempt is made to recognize a token. (If this `StreamTokenizer` is currently recognizing line terminators, then a line terminator that ends a comment will be recognized as a token in the same manner as any other line terminator in the contained input stream.)

If `/*` is encountered and this `StreamTokenizer` is currently recognizing `/*` comments, then all subsequent characters are skipped and ignored, up to and including the next occurrence of `*/` or end of file. Then another attempt is made to recognize a token.

If none of the cases listed above applies, then the only other possibility is that the first non-whitespace character encountered is an ordinary character. That character is considered to be a token and is stored in the `tttype` field and returned.

22.14.22 `public void pushBack()`

Calling this method "pushes back" the current token; that is, it causes the next call to `nextToken` to return the same token that it just provided. Note that this method does *not* restore the line number to its previous value, so if the method `lineno` is called after a call to `pushBack` but before the next call to `nextToken`, an incorrect line number may be returned.

22.14.23 `public int lineno()`

The number of the line on which the current token appeared is returned. The first token in the input stream, if not a line terminator, is considered to appear on line 1. A line terminator token is considered to appear on the line that it precedes, not on the line it terminates; thus, the first line terminator in the input stream is considered to be on line 2.

22.14.24 `public String toString()`

The current token and the current line number are converted to a string of the form:

```
"Token[x], line m"
```

where *m* is the current line number in decimal form and *x* depends on the type of the current token:

- If the token type is `TT_EOF`, then *x* is `"EOF"`.
- If the token type is `TT_EOL`, then *x* is `"EOL"`.

- If the token type is `TT_WORD`, then `x` is the current value of `sval` ([§22.14.6](#)).
- If the token type is `TT_NUMBER`, then `x` is `"n="` followed by the result of converting the current value of `nval` ([§22.14.7](#)) to a string ([§20.10.15](#)).

Overrides the `toString` method of `Object` ([§20.1.2](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.15 The Class java.io.OutputStream

An output stream accepts output bytes and sends them to some sink.

```
public abstract class OutputStream {
    public abstract void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void flush() throws IOException;
    public void close() throws IOException;
}
```

22.15.1 `public abstract void write(int b) throws IOException`

The general contract for `write` is that one byte is written to the output stream. The byte to be written is the eight low-order bits of the argument `b`. The 24 high-order bits of `b` are ignored.

If the byte cannot be written for any reason, an `IOException` is thrown. In particular, an `IOException` may be thrown if the output stream has been closed ([§22.15.5](#)).

22.15.2 `public void write(byte[] b)`
`throws IOException, NullPointerException`

The general contract for `write(b)` is that it should have exactly the same effect as the call `write(b, 0, b.length)` ([§22.15.3](#)).

The `write(b)` method for class `OutputStream` in fact makes such a call.

22.15.3 `public void write(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

The general contract for `write(b, off, len)` is that some of the bytes in the array `b` are written to the output stream as if one at a time, in order; element `b[off]` is the first byte written and `b[off+len-1]` is the last byte written by this operation.

If `b` is null, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

If the byte cannot be written for any reason, an `IOException` is thrown. In particular, an `IOException` is thrown if the output stream has been closed ([§22.15.5](#)).

The `write(b, off, len)` method for class `OutputStream` simply calls the method `write` ([§22.15.1](#)) repeatedly, once for each byte in `b` to be written.

22.15.4 `public void flush() throws IOException`

The general contract of `flush` is that calling it is an indication that, if any bytes previously written have been buffered by the implementation of the output stream, such bytes should immediately be written to their intended destination.

The `flush` method for class `OutputStream` does nothing and simply returns.

22.15.5 `public void close() throws IOException`

The general contract of `close` is that it closes the output stream. A closed stream cannot perform output operations and cannot be reopened.

The `close` method for class `OutputStream` does nothing and simply returns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.16 The Class java.io.FileOutputStream

A file output stream writes output bytes to a file in a file system. What files are available or may be created depends on the host environment.

```
public class FileOutputStream extends OutputStream {
    public FileOutputStream(String path)
        throws SecurityException, FileNotFoundException;
    public FileOutputStream(File file)
        throws SecurityException, FileNotFoundException;
    public FileOutputStream(FileDescriptor fdObj)
        throws SecurityException;
    public void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void close() throws IOException;
    public final FileDescriptor getFD() throws IOException;
    protected void finalize() throws IOException;
}
```

22.16.1 `public FileOutputStream(String path)`
 throws SecurityException, FileNotFoundException

This constructor initializes a newly created `FileOutputStream` by opening a connection to an actual file, the file named by the path name `path` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the `path` argument as its argument.

If the actual file cannot be opened, a `FileNotFoundException` is thrown.

22.16.2 `public FileOutputStream(File file)`
 throws SecurityException, FileNotFoundException

This constructor initializes a newly created `FileOutputStream` by opening a connection to an actual file, the file named by `file` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the path represented by the `file` argument as its argument.

If the actual file cannot be opened, a `FileNotFoundException` is thrown.

22.16.3 `public FileOutputStream(FileDescriptor fdObj)`
 throws SecurityException

This constructor initializes a newly created `FileOutputStream` by using the file descriptor `fdObj`, which represents an existing connection to an actual file in the file system.

First, if there is a security manager, its `checkWrite` method ([§20.17.20](#)) is called with the file descriptor `fdObj` argument as its argument.

22.16.4 `public final FileDescriptor getFD() throws IOException`

This method returns the `FileDescriptor` object ([§22.26](#)) that represents the connection to the actual file in the file system being used by this `FileOutputStream`.

22.16.5 `public void write(int b) throws IOException`

The byte for this operation is written to the actual file to which this file output stream is connected.

Implements the `write` method of `OutputStream` ([§22.15.1](#)).

22.16.6 `public void write(byte[] b)`
`throws IOException, NullPointerException`

Bytes for this operation are written to the actual file to which this file output stream is connected.

Overrides the `write` method of `OutputStream` ([§22.15.2](#)).

22.16.7 `public void write(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

Bytes for this operation are written to the actual file to which this file output stream is connected.

Overrides the `write` method of `OutputStream` ([§22.15.3](#)).

22.16.8 `public void close() throws IOException`

This file output stream is closed and may no longer be used for writing bytes.

Overrides the `close` method of `OutputStream` ([§22.15.5](#)).

22.16.9 `protected void finalize() throws IOException`

A `FileOutputStream` uses finalization to clean up the connection to the actual file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.17 The Class java.io.PipedOutputStream

A piped output stream should be connected to a piped input stream; the piped input stream then provides whatever data bytes are written to the piped output stream. Typically, data is written to a `PipedOutputStream` object by one thread and data is read from the corresponding `PipedInputStream` (§22.5) by some other thread. Attempting to use both objects from a single thread is not recommended, as it may deadlock the thread.

```
public class PipedOutputStream extends OutputStream {
    public PipedOutputStream(PipedInputStream snk)
        throws IOException;
    public PipedOutputStream();
    public void connect(PipedInputStream snk)
        throws IOException;
    public void write(int b) throws IOException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void close() throws IOException;
}
```

22.17.1 `public PipedOutputStream(PipedInputStream snk)`
throws `IOException`

This constructor initializes a newly created `PipedOutputStream` so that it is connected to the piped input stream `snk`. Data bytes written to this stream will then be available as input from `snk`.

22.17.2 `public PipedOutputStream()`

This constructor initializes a newly created `PipedOutputStream` so that it is not yet connected. It must be connected to a `PipedInputStream` before being used.

22.17.3 `public void connect(PipedInputStream snk)`
throws `IOException`

The `connect` method causes this piped output stream to be connected to the piped input stream `snk`. If this object is already connected to some other piped input stream, an `IOException` is thrown.

If `snk` is an unconnected piped input stream and `src` is an unconnected piped output stream, they may be connected by either the call:

```
src.connect(snk)
```

or the call:

```
snk.connect(src)
```


The two calls have the same effect.

22.17.4 `public void write(int b) throws IOException`

If a thread was reading data bytes from the connected piped input stream, but the thread is no longer alive, then an `IOException` is thrown.

Implements the `write` method of `OutputStream` ([§22.15.1](#)).

22.17.5 `public void write(byte[] b, int off, int len)`
`throws IOException, NullPointerException,`
`IndexOutOfBoundsException`

If a thread was reading data bytes from the connected piped input stream, but the thread is no longer alive, then an `IOException` is thrown.

Overrides the `write` method of `OutputStream` ([§22.15.3](#)).

22.17.6 `public void close() throws IOException`

This piped output stream is closed and may no longer be used for writing bytes.

Overrides the `close` method of `OutputStream` ([§22.15.5](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.18 The Class java.io.ByteArrayOutputStream

A `ByteArrayOutputStream` contains an internal buffer that accumulates all the bytes written to the stream since its creation or the most recent call to the `reset` method. At any point, the bytes written to the stream so far may be retrieved in the form of an array of bytes or a `String`. The bytes written so far may also be copied to some other output stream. The `size` method returns the number of characters written so far.

```
public class ByteArrayOutputStream extends OutputStream {
    protected byte[] buf;
    protected int count;
    public ByteArrayOutputStream();
    public ByteArrayOutputStream(int size);
    public void write(int b);
    public void write(byte[] b, int off, int len)
        throws NullPointerException, IndexOutOfBoundsException;
    public int size();
    public void reset();
    public byte[] toByteArray();
    public String toString();
    public String toString(int hibyte);
    public void writeTo(OutputStream out) throws IOException;
}
```

22.18.1 `protected byte[] buf;`

An internal array of bytes. Elements `buf[0]` through `buf[count-1]` are the bytes that have been written to the stream since its creation or the last `reset` (§22.18.8) operation.

22.18.2 `protected int count;`

This value should always be nonnegative. It is the number of bytes that have been written to the stream since its creation or the last `reset` (§22.18.8) operation.

22.18.3 `public ByteArrayOutputStream()`

This constructor initializes a newly created `ByteArrayOutputStream` so that its internal buffer array has length 32.

22.18.4 `public ByteArrayOutputStream(int size)`

This constructor initializes a newly created `ByteArrayOutputStream` so that its internal buffer array has length `size`. This matters only for reasons of efficiency; the buffer array is replaced by a larger one whenever necessary to accommodate additional bytes written to the stream.

22.18.5 `public void write(int b)`

One byte is added on the internal buffer. The byte to be added is the eight low-order bits of the

argument `n`. The 24 high-order bits of `n` are ignored.

Implements the `write` method of `OutputStream` (§22.15.1).

22.18.6 `public void write(byte[] b, int off, int len)`
throws `NullPointerException`, `IndexOutOfBoundsException`

Elements `b[off]` through `b[off+len-1]` are appended to the internal buffer.

If `b` is `null`, a `NullPointerException` is thrown.

If `off` is negative, or `len` is negative, or `off+len` is greater than the length of the array `b`, then an `IndexOutOfBoundsException` is thrown.

Overrides the `write` method of `OutputStream` (§22.15.3).

22.18.7 `public int size()`

The current value of `count` is returned.

22.18.8 `public void reset()`

The internal variable `count` is reset to zero, thereby logically discarding all bytes written to the stream so far. However, the internal buffer array, which may be quite large, remains as it is.

22.18.9 `public byte[] toByteArray()`

A new array of bytes is created and returned. Its length is equal to the current value of `count`. Its initial contents are copies of the bytes written to the stream so far—that is, elements 0 through `count-1` of `buf`.

22.18.10 `public String toString()`

A new `String` is created and returned. Its length is equal to the current value of `count`. Its initial contents are copies of the bytes written to the stream so far—that is, elements 0 through `count-1` of `buf`, zero-extended to produce characters. Thus, `toString()` has the same effect as `toString(0)` (§22.18.11).

Overrides the `toString` method of `Object` (§20.1.2).

22.18.11 `public String toString(int hiByte)`

A new array of bytes is created and returned. Its length is equal to the current value of `count`. Its initial contents are copies of the bytes written to the stream so far—that is, elements 0 through `count-1` of `buf`—with `hiByte` supplying the high-order eight bits of each character. Thus, character `k` of the result is equal to:

```
((hiByte & 0xff) << 8) | (buf[k] & 0xff)
```

See the `String` constructor that accepts a `hiByte` argument (§20.12.6).

22.18.12 `public void writeTo(OutputStream out) throws IOException`

The current contents of the internal buffer are written to the output stream `out` by the call:

```
out.write(buf, 0, count)
```

Note that if `out` is the same as `this`, the effect is simply to append to the buffer a copy of its current contents, thereby doubling the number of buffered bytes. This may not be a particularly useful effect; the point is merely that the operation does terminate, having had a sensible effect, rather than running off into an endless loop.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.19 The Class java.io.FilterOutputStream

A `FilterOutputStream` contains some other output stream, which it uses as its basic sink of data, possibly transforming the data along the way or providing additional functionality. The class `FilterOutputStream` itself simply overrides all methods of `OutputStream` with versions that pass all requests to the contained output stream. Subclasses of `FilterOutputStream` may further override some of these methods and may also provide additional methods and fields.

```
public class FilterOutputStream extends OutputStream {
    protected OutputStream out;
    public FilterOutputStream(OutputStream out);
    public void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void flush() throws IOException;
    public void close() throws IOException;
}
```

22.19.1 `protected OutputStream out;`

The output stream to be filtered.

22.19.2 `public FilterOutputStream(OutputStream out)`

This constructor initializes a newly created `FilterInputStream` by assigning the argument `out` to the field `this.out` so as to remember it for later use.

22.19.3 `public void write(int b) throws IOException`

This method simply performs `out.write(b)`.

Implements the abstract `write` method of `OutputStream` ([§22.15.1](#)).

22.19.4 `public void write(byte[] b)`
`throws IOException, NullPointerException`

This method simply performs `out.write(b)`.

Overrides the `write` method of `OutputStream` ([§22.15.2](#)).

22.19.5 `public void write(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

This method simply performs `out.write(b, off, len)`.

Overrides the `write` method of `OutputStream` ([§22.15.3](#)).

22.19.6 `public void flush() throws IOException`

This method simply performs `out.flush()`.

Overrides the `flush` method of `OutputStream` ([§22.15.4](#)).

22.19.7 `public void close() throws IOException`

This method simply performs `out.close()`.

Overrides the `close` method of `OutputStream` ([§22.15.5](#)).

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

22.20 The Class java.io.BufferedOutputStream

A `BufferedOutputStream` adds functionality to another output stream, namely the ability to buffer the output. When the `BufferedOutputStream` is created, an internal buffer array is created. As bytes are written to the stream, they are stored in the internal buffer, which is flushed as necessary, thereby performing output to the contained output stream in large blocks rather than a byte at a time.

```
public class BufferedOutputStream extends FilterOutputStream {
    protected byte[] buf;
    protected int count;
    public BufferedOutputStream(OutputStream out);
    public BufferedOutputStream(OutputStream out, int size);
    public void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void flush() throws IOException;
}
```

22.20.1 `protected byte[] buf;`

The internal buffer array.

22.20.2 `protected int count;`

This value is always in the range 0 through `buf.length`; elements `buf[0]` through `buf[count-1]` contain valid byte data.

22.20.3 `public BufferedOutputStream(OutputStream out)`

This constructor initializes a newly created `BufferedOutputStream` by saving its argument, the input stream `out`, for later use. An internal buffer array is created and stored in `buf`.

22.20.4 `public BufferedOutputStream(OutputStream out, int size)`

This constructor initializes a newly created `BufferedOutputStream` by saving its argument, the input stream `out`, for later use. An internal buffer array of length `size` is created and stored in `buf`.

22.20.5 `public void write(int b) throws IOException`

See the general contract of the `write` method of `OutputStream` ([§22.15.1](#)).

Overrides the `write` method of `FilterOutputStream` ([§22.19.3](#)).

22.20.6 `public void write(byte[] b)`
`throws IOException, NullPointerException`

See the general contract of the `write` method of `OutputStream` ([§22.15.2](#)).

Overrides the `write` method of `FilterOutputStream` ([§22.19.4](#)).

22.20.7 `public void write(byte[] b, int off, int len)`
throws `IOException`, `NullPointerException`, `IndexOutOfBoundsException`

See the general contract of the `write` method of `OutputStream` ([§22.15.3](#)).

Overrides the `write` method of `FilterOutputStream` ([§22.19.5](#)).

22.20.8 `public void flush()` throws `IOException`

See the general contract of the `flush` method of `OutputStream` ([§22.15.4](#)).

Overrides the `flush` method of `FilterOutputStream` ([§22.19.6](#)).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

22.21 The Class java.io.DataOutputStream

A data output stream provides facilities for converting data of diverse types into character sequence of specific formats that are then sent to some output stream.

```
public class DataOutputStream extends FilterOutputStream
    implements DataOutput {
    protected int written;
    public DataOutputStream(OutputStream out);
    public void write(int b) throws IOException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public void flush() throws IOException;
    public final void writeBoolean(boolean v) throws IOException;
    public final void writeByte(int v) throws IOException;
    public final void writeShort(int v) throws IOException;
    public final void writeChar(int v) throws IOException;
    public final void writeInt(int v) throws IOException;
    public final void writeLong(long v) throws IOException;
    public final void writeFloat(float v) throws IOException;
    public final void writeDouble(double v) throws IOException;
    public final void writeBytes(String s)
        throws IOException, NullPointerException;
    public final void writeChars(String s)
        throws IOException, NullPointerException;
    public final void writeUTF(String str)
        throws IOException, NullPointerException;
    public final int size();
}
```

22.21.1 protected int written;

This field contains the number of bytes written to the stream so far.

22.21.2 public DataOutputStream(OutputStream out)

This constructor initializes a newly created `DataOutputStream` by saving its argument, the output stream `out`, for later use. The counter `written` is set to zero.

22.21.3 public void write(int b) throws IOException

The byte for this operation (the low eight bits of the argument `b`) is written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 1.

Implements the `write` method of `OutputStream` ([§22.15.1](#)).

22.21.4 public void write(byte[] b, int off, int len)
throws IOException, NullPointerException, IndexOutOfBoundsException

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by `len`.

Overrides the `write` method of `OutputStream` ([§22.15.3](#)).

22.21.5 `public void flush() throws IOException`

The contained output stream is flushed.

Overrides the `flush` method of `OutputStream` ([§22.15.4](#)).

22.21.6 `public final void writeBoolean(boolean v)`
`throws IOException`

See the general contract of the `writeBoolean` method of `DataOutput` ([§22.2.4](#)).

The byte for this operation is written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 1.

22.21.7 `public final void writeByte(int v) throws IOException`

See the general contract of the `writeByte` method of `DataOutput` ([§22.2.5](#)).

The byte for this operation is written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 1.

22.21.8 `public final void writeShort(int v) throws IOException`

See the general contract of the `writeShort` method of `DataOutput` ([§22.2.6](#)).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 2.

22.21.9 `public final void writeChar(int v) throws IOException`

See the general contract of the `writeChar` method of `DataOutput` ([§22.2.7](#)).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 2.

22.21.10 `public final void writeInt(int v) throws IOException`

See the general contract of the `writeInt` method of `DataOutput` ([§22.2.8](#)).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 4.

22.21.11 `public final void writeLong(long v) throws IOException`

See the general contract of the `writeLong` method of `DataOutput` ([§22.2.9](#)).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 8.

22.21.12 `public final void writeFloat(float v) throws IOException`

See the general contract of the `writeFloat` method of `DataOutput` (§22.2.10).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 4.

22.21.13 `public final void writeDouble(double v) throws IOException`

See the general contract of the `writeDouble` method of `DataOutput` (§22.2.11).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by 8.

22.21.14 `public final void writeBytes(String s)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `writeBytes` method of `DataOutput` (§22.2.12).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by the length of `s`.

22.21.15 `public final void writeChars(String s)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `writeChars` method of `DataOutput` (§22.2.13).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by twice the length of `s`.

22.21.16 `public final void writeUTF(String str)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `writeUTF` method of `DataOutput` (§22.2.14).

Bytes for this operation are written to the contained output stream. If no exception is thrown, the counter `written` is incremented by the total number of bytes written to the output stream. This will be at least two plus the length of `s`, and at most two plus thrice the length of `s`.

22.21.17 `public final int size()`

The `size` method returns the current value of the counter `written`, the number of bytes written to the stream so far.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.22 The Class java.io.PrintStream

A `PrintStream` adds functionality to another output stream—namely, the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a `PrintStream` never throws an `IOException`; instead, exceptional situations merely set an internal flag that can be tested by the `checkError` method. Optionally, a `PrintStream` can be created so as to "autoflush"; this means that after an array of bytes is written, or after a single byte equal to `'\n'` is written, the `flush` method is automatically invoked.

```
public class PrintStream extends FilterOutputStream {
    public PrintStream(OutputStream out);
    public PrintStream(OutputStream out, boolean autoflush);
    public void write(int b);
    public void write(byte[] b, int off, int len)
        throws NullPointerException, IndexOutOfBoundsException;
    public void flush();
    public void close();
    public boolean checkError();
    public void print(Object obj);
    public void print(String s);
    public void print(char[] s) throws NullPointerException;
    public void print(char c);
    public void print(int i);
    public void print(long l);
    public void print(float f);
    public void print(double d);
    public void print(boolean b);
    public void println();
    public void println(Object obj);
    public void println(String s);
    public void println(char[] s) throws NullPointerException;
    public void println(char c);
    public void println(int i);
    public void println(long l);
    public void println(float f);
    public void println(double d);
    public void println(boolean b);
}
```

22.22.1 `public PrintStream(OutputStream out)`

This constructor initializes a newly created `PrintStream` by saving its argument, the output stream `out`, for later use. This stream will not autoflush.

22.22.2 `public PrintStream(OutputStream out, boolean autoflush)`

This constructor initializes a newly created `PrintStream` by saving its argument, the output stream `out`, for later use. This stream will autoflush if and only if `autoflush` is `true`.

22.22.3 `public void write(int b)`

See the general contract of the `write` method of `OutputStream` (§22.15.1).

Overrides the `write` method of `FilterOutputStream` (§22.19.3).

```
22.22.4      public void write(byte[] b, int off, int len)
               throws NullPointerException,      IndexOutOfBoundsException
```

See the general contract of the `write` method of `OutputStream` (§22.15.3).

Overrides the `write` method of `FilterOutputStream` (§22.19.5).

```
22.22.5      public void flush()
```

See the general contract of the `flush` method of `OutputStream` (§22.15.4).

Overrides the `flush` method of `FilterOutputStream` (§22.19.6).

```
22.22.6      public void close()
```

See the general contract of the `close` method of `OutputStream` (§22.15.5).

Overrides the `close` method of `FilterOutputStream` (§22.19.7).

```
22.22.7      public boolean checkError()
```

The result is `true` if and only if this output stream has ever encountered any kind of trouble—that is, if any operation on the contained output stream has ever resulted in an `IOException` other than an `InterruptedException`. If an operation on the contained output stream throws an `InterruptedException`, then the `PrintStream` class converts the exception back to an interrupt by doing:

```
Thread.currentThread().interrupt();
```

or the equivalent.

```
22.22.8      public void print(Object obj)
```

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(obj)` (§20.12.38) are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

```
22.22.9      public void print(String s)
```

The low-order bytes of the characters in the string `s` are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3). If `s` is `null`, then the low-order bytes of the four characters `n`, `u`, `l`, `l` are written to the contained output stream.

```
22.22.10     public void print(char[] s) throws NullPointerException
```


The low-order bytes of the characters in the character array `s` are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

If `s` is `null`, a `NullPointerException` is thrown.

22.22.11 `public void print(boolean b)`

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(b)` (§20.12.41) as a string are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.12 `public void print(char c)`

The low-order byte of the character `c` is written to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.13 `public void print(int i)`

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(i)` (§20.12.43) as a string are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.14 `public void print(long l)`

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(l)` (§20.12.44) as a string are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.15 `public void print(float f)`

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(f)` (§20.12.45) as a string are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.16 `public void print(double d)`

The low-order bytes of the characters in the `String` that would be produced by `String.valueOf(d)` (§20.12.46) as a string are written, in order, to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.17 `public void println()`

The low-order byte of the newline character `'\n'` is written to the contained output stream in exactly the manner of the `write` method (§22.22.3).

22.22.18 `public void println(Object obj)`

This is exactly the same as `print(obj)` (§22.22.8) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.19 `public void println(String s)`

This is exactly the same as `print(s)` (§22.22.9) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.20 `public void println(char[] s) throws NullPointerException`

This is exactly the same as `print(s)` (§22.22.10) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

If `s` is null, a `NullPointerException` is thrown.

22.22.21 `public void println(boolean b)`

This is exactly the same as `print(b)` (§22.22.11) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.22 `public void println(char c)`

This is exactly the same as `print(c)` (§22.22.12) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.23 `public void println(int i)`

This is exactly the same as `print(i)` (§22.22.13) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.24 `public void println(long l)`

This is exactly the same as `print(l)` (§22.22.14) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.25 `public void println(float f)`

This is exactly the same as `print(f)` (§22.22.15) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

22.22.26 `public void println(double d)`

This is exactly the same as `print(d)` (§22.22.16) followed by writing the low-order byte of the newline character `'\n'` to the contained output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.23 The Class java.io.RandomAccessFile

A random access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the *file pointer*; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended. The file pointer can be read by the `getFilePointer` method and set by the `seek` method.

```
public class RandomAccessFile implements DataOutput, DataInput {
    public RandomAccessFile(String path, String mode)
        throws SecurityException, IOException,
            IllegalArgumentException;
    public RandomAccessFile(File file, String mode)
        throws SecurityException, IOException,
            IllegalArgumentException;
    public final FileDescriptor getFD() throws IOException;
    public native long getFilePointer() throws IOException;
    public native void seek(long pos) throws IOException;
    public native long length() throws IOException;
    public native void close() throws IOException;
    public native int read() throws IOException;
    public int read(byte[] b)
        throws IOException, NullPointerException;
    public int read(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    // The methods that implement interface DataInput:
    public final void readFully(byte[] b)
        throws IOException, NullPointerException;
    public final void readFully(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
    public int skipBytes(int n) throws IOException;
    public final boolean readBoolean() throws IOException;
    public final byte readByte() throws IOException;
    public final int readUnsignedByte() throws IOException;
    public final short readShort() throws IOException;
    public final int readUnsignedShort() throws IOException;
    public final char readChar() throws IOException;
    public final int readInt() throws IOException;
    public final long readLong() throws IOException;
    public final float readFloat() throws IOException;
    public final double readDouble() throws IOException;
    public final String readLine() throws IOException;
    public final String readUTF() throws IOException;
    // The methods that implement interface DataOutput:
    public native void write(int b) throws IOException;
    public void write(byte[] b)
        throws IOException, NullPointerException;
    public void write(byte[] b, int off, int len)
        throws IOException, NullPointerException,
            IndexOutOfBoundsException;
```



```

    public final void writeBoolean(boolean v) throws IOException;
    public final void writeByte(int v) throws IOException;
    public final void writeShort(int v) throws IOException;
    public final void writeChar(int v) throws IOException;
    public final void writeInt(int v) throws IOException;
    public final void writeLong(long v) throws IOException;
    public final void writeFloat(float v) throws IOException;
    public final void writeDouble(double v) throws IOException;
    public final void writeBytes(String s) throws IOException;
    public final void writeChars(String s) throws IOException;
    public final void writeUTF(String str) throws IOException;
}

```

It is generally true of all the reading routines in this class that if end of file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end of file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the stream has been closed ([§22.23.7](#)).

22.23.1 `public RandomAccessFile(String path, String mode)`
throws `SecurityException`, `IOException`, `IllegalArgumentException`

This constructor initializes a newly created `RandomAccessFile` by opening a connection to an actual file, the file named by the path name `path` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the `path` argument as its argument.

Next, if `mode` is "rw" and there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the `path` argument as its argument.

If `mode` is "rw", then the file may be both read and written. If `mode` is "r", then the file may be read but may not be written (every write method for this object will simply throw an `IOException`). If `mode` is not "r" or "rw", then this constructor throws an `IllegalArgumentException`.

22.23.2 `public RandomAccessFile(File file, String mode)`
throws `SecurityException`, `IOException`, `IllegalArgumentException`

This constructor initializes a newly created `RandomAccessFile` by opening a connection to an actual file, the file named by `file` in the file system. A new `FileDescriptor` object is created to represent this file connection.

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by the `file` argument as its argument.

Next, if `mode` is "rw" and there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the path represented by the `file` argument as its argument.

If `mode` is "rw", then the file may be both read and written. If `mode` is "r", then the file may be read but may not be written (every write method for this object will simply throw an `IOException`). If `mode` is not "r" or "rw", then this constructor throws an `IllegalArgumentException`.

22.23.3 `public final FileDescriptor getFD() throws IOException`

This method returns the `FileDescriptor` object (§22.26) that represents the connection to the actual file in the file system being used by this `RandomAccessFile`.

22.23.4 `public long getFilePointer() throws IOException`

The current file pointer for this random access file is returned. An `IOException` is thrown if the file pointer cannot be read for any reason.

22.23.5 `public void seek(long pos) throws IOException`

The file pointer for this random access file is set to `pos`, which is a position within the file, measured in bytes. Position 0 is the start of the file. An `IOException` is thrown if `pos` is less than zero or greater than the length of the file, or if the file pointer cannot be set for any other reason.

22.23.6 `public long length() throws IOException`

The length of this random access file, measured in bytes, is returned.

An `IOException` is thrown if the length cannot be read for any reason.

22.23.7 `public void close() throws IOException`

This random access file is closed. A closed random access file cannot perform input or output operations and cannot be reopened.

22.23.8 `public int read() throws IOException`

This method reads one byte from the random access file. The byte is returned as an integer in the range 0 to 255 (0x00-0xff). If no byte is available because the file pointer is at end of file, the value -1 is returned.

If the byte cannot be read for any reason other than end of file, an `IOException` is thrown. In particular, an `IOException` is thrown if the input stream has been closed (§22.23.7).

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `read` method of `InputStream` (§22.3.1).

22.23.9 `public int read(byte[] b)`
`throws IOException, NullPointerException`

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `read` method of `InputStream` (§22.3.2).

22.23.10 `public int read(byte[] b, int off, int len)`
`throws IOException, NullPointerException, IndexOutOfBoundsException`

Although `RandomAccessFile` is not a subclass of `InputStream`, this method behaves in exactly the same way as the `read` method of `InputStream` (§22.3.3).

22.23.11 `public final void readFully(byte[] b)`

throws IOException, NullPointerException

See the general contract of the `readFully` method of `DataInput` ([§22.1.1](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.12 `public final void readFully(byte[] b, int off, int len) throws
IOException, NullPointerException, IndexOutOfBoundsException`

See the general contract of the `readFully` method of `DataInput` ([§22.1.2](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.13 `public int skipBytes(int n) throws IOException`

See the general contract of the `skipBytes` method of `DataInput` ([§22.1.3](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.14 `public final boolean readBoolean() throws IOException`

See the general contract of the `readBoolean` method of `DataInput` ([§22.1.4](#)).

The byte for this operation is read from the random access file, starting at the current file pointer.

22.23.15 `public final byte readByte() throws IOException`

See the general contract of the `readByte` method of `DataInput` ([§22.1.5](#)).

The byte for this operation is read from the random access file, starting at the current file pointer.

22.23.16 `public final int readUnsignedByte() throws IOException`

See the general contract of the `readUnsignedByte` method of `DataInput` ([§22.1.6](#)).

The byte for this operation is read from the random access file, starting at the current file pointer.

22.23.17 `public final short readShort() throws IOException`

See the general contract of the `readShort` method of `DataInput` ([§22.1.7](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.18 `public final int readUnsignedShort() throws IOException`

See the general contract of the `readUnsignedShort` method of `DataInput` ([§22.1.8](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.19 `public final char readChar() throws IOException`

See the general contract of the `readChar` method of `DataInput` ([§22.1.9](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.20 `public final int readInt() throws IOException`

See the general contract of the `readInt` method of `DataInput` ([§22.1.10](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.21 `public final long readLong() throws IOException`

See the general contract of the `readLong` method of `DataInput` ([§22.1.11](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.22 `public final float readFloat() throws IOException`

See the general contract of the `readFloat` method of `DataInput` ([§22.1.12](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.23 `public final double readDouble() throws IOException`

See the general contract of the `readDouble` method of `DataInput` ([§22.1.13](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.24 `public final String readLine() throws IOException`

See the general contract of the `readLine` method of `DataInput` ([§22.1.14](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.25 `public final String readUTF() throws IOException`

See the general contract of the `readUTF` method of `DataInput` ([§22.1.15](#)).

Bytes for this operation are read from the random access file, starting at the current file pointer.

22.23.26 `public void write(int b) throws IOException;`

See the general contract of the `write` method of `DataOutput` ([§22.2.1](#)).

The byte for this operation is written to the random access file, starting at the current file pointer.

22.23.27 `public void write(byte[] b)`
`throws IOException, NullPointerException`

See the general contract of the `write` method of `DataOutput` ([§22.2.2](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.28 `public void write(byte[] b, int off, int len)`

throws IOException, NullPointerException, IndexOutOfBoundsException

See the general contract of the `write` method of `DataOutput` ([§22.2.3](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.29 `public final void writeBoolean(boolean v)`
throws IOException

See the general contract of the `writeBoolean` method of `DataOutput` ([§22.2.4](#)).

The byte for this operation is written to the random access file, starting at the current file pointer.

22.23.30 `public final void writeByte(int v) throws IOException`

See the general contract of the `writeByte` method of `DataOutput` ([§22.2.5](#)).

The byte for this operation is written to the random access file, starting at the current file pointer.

22.23.31 `public final void writeShort(int v) throws IOException`

See the general contract of the `writeShort` method of `DataOutput` ([§22.2.6](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.32 `public final void writeChar(int v) throws IOException`

See the general contract of the `writeChar` method of `DataOutput` ([§22.2.7](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.33 `public final void writeInt(int v) throws IOException`

See the general contract of the `writeInt` method of `DataOutput` ([§22.2.8](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.34 `public final void writeLong(long v) throws IOException`

See the general contract of the `writeLong` method of `DataOutput` ([§22.2.9](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.35 `public final void writeFloat(float v) throws IOException`

See the general contract of the `writeFloat` method of `DataOutput` ([§22.2.10](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.36 `public final void writeDouble(double v)`
throws IOException

See the general contract of the `writeDouble` method of `DataOutput` ([§22.2.11](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.37 `public final void writeBytes(String s) throws IOException`

See the general contract of the `writeBytes` method of `DataOutput` ([§22.2.12](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.38 `public final void writeChars(String s) throws IOException`

See the general contract of the `writeChars` method of `DataOutput` ([§22.2.13](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

22.23.39 `public final void writeUTF(String str) throws IOException`

See the general contract of the `writeUTF` method of `DataOutput` ([§22.2.14](#)).

Bytes for this operation are written to the random access file, starting at the current file pointer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.24 The Class java.io.File

A `File` object contains a *path*, which is a character string that can be used to identify a file within a file system. A path is assumed to consist of two parts, the *directory* and the *file name*, separated by the last occurrence within the path of a particular character known as the *separator character*. Some methods provide access to parts of the path string; other methods operate on the file that is identified by the path string. The details of such operations on files are to some extent dependent on the implementation of the host file system. The `File` class is designed to provide a set of abstract operations that are reasonably portable across otherwise incompatible file systems.

```
public class File {
    public static final String separator =
        System.getProperty("file.separator");
    public static final char separatorChar =
        separator.charAt(0);
    public static final String pathSeparator =
        System.getProperty("path.separator");
    public static final char pathSeparatorChar =
        pathSeparator.charAt(0);
    public File(String path) throws NullPointerException;
    public File(String dirname, String name)
        throws NullPointerException;
    public File(File dir, String name)
        throws NullPointerException;
    public String toString();
    public boolean equals(Object obj);
    public int hashCode();
    public String getName();
    public String getPath();
    public String getAbsolutePath();
    public String getParent();
    public native boolean isAbsolute();
    public boolean exists() throws SecurityException;
    public boolean canRead() throws SecurityException;
    public boolean canWrite() throws SecurityException;
    public boolean isFile() throws SecurityException;
    public boolean isDirectory() throws SecurityException;
    public long lastModified() throws SecurityException;
    public long length() throws SecurityException;
    public boolean mkdir() throws SecurityException;
    public boolean mkdirs() throws SecurityException;
    public String[] list() throws SecurityException;
    public String[] list(FilenameFilter filter)
        throws SecurityException;
    public boolean delete() throws SecurityException;
    public boolean renameTo(File dest) throws SecurityException;
}
```

22.24.1 `public static final String separator =`
`System.getProperty("file.separator");`

This string should consist of a single character, whose value is also available in the field

separatorChar; the string is provided merely for convenience.

22.24.2 `public static final char separatorChar = separator.charAt(0);`

The last occurrence of this character in a path string is assumed to separate the directory part of the path from the file name part of the path. On UNIX systems this character is typically `'/'`.

22.24.3 `public static final String pathSeparator =
System.getProperty("path.separator");`

This string should consist of a single character, whose value is also available in the field `pathSeparatorChar`; the string is provided merely for convenience.

22.24.4 `public static final char pathSeparatorChar =
pathSeparator.charAt(0);`

The first occurrence of this character in a string is sometimes assumed to separate a host name from a path name. On UNIX systems this character is typically `':'`.

22.24.5 `public File(String path) throws NullPointerException`

This constructor initializes a newly created `File` so that it represents the path indicated by the argument `path`.

If the `path` is null, a `NullPointerException` is thrown.

22.24.6 `public File(String dirname, String name)
throws NullPointerException`

This constructor initializes a newly created `File` so that it represents the path whose directory part is specified by the argument `dirname` and whose file name part is specified by the argument `name`. If the `dirname` argument is null, the `name` is used as the path; otherwise the concatenation of `dirname`, the `separatorChar` (§22.24.2), and the `name` is used as the path.

If the `name` is null, a `NullPointerException` is thrown.

22.24.7 `public File(File dir, String name)
throws NullPointerException`

This constructor initializes a newly created `File` so that it represents the path whose directory part is specified by the `File` object `dir` and whose file name part is specified by the argument `name`.

If the `name` is null, a `NullPointerException` is thrown.

22.24.8 `public String toString()`

The result is a `String` equal to the path represented by this `File` object.

Overrides the `toString` method of `Object` (§20.1.2).

22.24.9 `public boolean equals(Object obj)`

The result is `true` if and only if the argument is not `null` and is a `File` object that represents the same path as this `File` object. In other words, two `File` objects are equal if and only if the strings returned by the `getPath` method (§22.24.12) are equal.

Overrides the `equals` method of `Object` (§20.1.3).

22.24.10 `public int hashCode()`

The hash code of this `File` object is equal to the exclusive OR of the hash code of its path string and the decimal value 1234321:

```
this.getPath().hashCode() ^ 1234321
```

Overrides the `hashCode` method of `Object` (§20.1.4).

22.24.11 `public String getName()`

If the path string contains the `separatorChar` character (§22.24.2), this method returns the substring of the path that follows the last occurrence of the separator character; otherwise, the entire path string is returned.

22.24.12 `public String getPath()`

The result is a `String` equal to the path represented by this `File` object.

22.24.13 `public String getAbsolutePath()`

The result is a `String` equal to the result of converting to "absolute form" the path represented by this `File` object.

22.24.14 `public String getParent()`

If the path has a parent directory, a `String` representing the path of that parent directory is returned; otherwise, `null` is returned.

22.24.15 `public boolean isAbsolute()`

The result is `true` if and only if the path represented by the `File` object is in absolute form, indicating a complete name that starts from the root of the directory hierarchy, rather than a name relative to some implied directory.

22.24.16 `public boolean exists() throws SecurityException`

First, if there is a security manager, its `checkRead` method (§20.17.19) is called with the path represented by this `File` object as its argument.

The result is `true` if and only if the file system actually contains a file that is specified by the path of the `File` object.

22.24.17 `public boolean canRead() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

The result is `true` if and only if both of the following are true:

- The file system actually contains a file specified by the path of the `File` object.
- The file so specified can be read.

22.24.18 `public boolean canWrite() throws SecurityException`

First, if there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the path represented by this `File` object as its argument.

The result is `true` if and only if both of the following are true:

- The file system actually contains a file specified by the path of the `File` object.
- The file so specified can be written.

22.24.19 `public boolean isFile() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

The result is `true` if and only if both of the following are true:

- The file system actually contains a file specified by the path of the `File` object.
- The file so specified is a data file rather than a directory.

22.24.20 `public boolean isDirectory() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

The result is `true` if and only if both of the following are true:

- The file system actually contains a file specified by the path of the `File` object.
- The file so specified is a directory rather than a data file.

22.24.21 `public long lastModified() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

An abstract modification time is returned. If two values returned by this method are compared, whether for the same file or for two different files, the smaller value represents an earlier modification time. Abstract modification times do not necessarily bear any relationship, even monotonicity, to times returned by the method `System.currentTimeMillis` ([§20.18.6](#)).

22.24.22 `public long length() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

The length of the file, measured in bytes, is returned.

22.24.23 `public boolean mkdir() throws SecurityException`

First, if there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called with the path represented by this `File` object as its argument.

An attempt is made to create the directory specified by the path represented by this `File` object; the result is `true` if and only if the creation operation succeeds.

22.24.24 `public boolean mkdirs() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

If the directory name represented by this `File` object has a parent directory name ([§22.24.14](#)), an attempt is first made to create the parent directory; if this attempt fails, the result is `false`.

Otherwise, once the parent directory has been determined to exist, or if the path has no parent, an attempt is made to create the directory specified by this `File` object. The result is `true` if and only if the creation operation succeeds.

22.24.25 `public String[] list() throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

If the path represented by this `File` object does not correspond to a directory in the file system, then `null` is returned. Otherwise, an array of strings is returned, one for each file in the directory (on UNIX systems, the names `"."` and `".."` are not included). Each string is a file name, not a complete path. There is no guarantee that the strings will appear in any particular order within the array; for example, they are not guaranteed to appear in alphabetical order.

22.24.26 `public String[] list(FilenameFilter filter)`
`throws SecurityException`

First, if there is a security manager, its `checkRead` method ([§20.17.19](#)) is called with the path represented by this `File` object as its argument.

If the path represented by this `File` object does not correspond to a directory in the file system, then `null` is returned. Otherwise, an array of strings is returned, one for each file in the directory (on UNIX systems, the names `"."` and `".."` are not included) whose name satisfies the given `filter`. Each string is a file name, not a complete path. There is no guarantee that the strings will appear in any particular order within the array; for example, they are not guaranteed to appear in alphabetical order. A file name satisfies the filter if and only if the value `true` results when the `accept` method ([§22.25.1](#)) of the filter is called with this `File` object and the name as arguments.

22.24.27 `public boolean renameTo(File dest)`
throws `SecurityException`

First, if there is a security manager, its `checkWrite` method ([§20.17.21](#)) is called twice, first with the path represented by this `File` object as its argument and again with the path of `dest` as its argument.

An attempt is made to rename the file specified by the path represented by this `File` object to the name specified by `dest`; the result is `true` if and only if the renaming operation succeeds.

22.24.28 `public boolean delete() throws SecurityException`

First, if there is a security manager, its `checkDelete` method ([§20.17.22](#)) is called with the path represented by this `File` object as its argument.

An attempt is made to delete the file specified by the path represented by this `File` object; the result is `true` if and only if the deletion operation succeeds.

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

22.25 The Interface java.io.FilenameFilter

The `list` method ([§22.24.26](#)) of class `File` requires, as an argument, an object that implements the `FilenameFilter` interface. The only purpose of such an object is to provide a method `accept` that decides which files should appear in the generated directory listing.

```
public interface FilenameFilter {  
    public boolean accept(File dir, String name);  
}
```

22.25.1 `public boolean accept(File dir, String name)`

This method should return `true` if and only if the given file named `name` in the directory `dir` is to appear in the final list of files generated by the `list` method ([§22.24.26](#)) of class `File`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.26 The Class java.io.FileDescriptor

A `FileDescriptor` is an opaque representation of a connection to an actual file in a file system, or to a network socket, or to another source or sink of bytes. The main practical use for a file descriptor is to create a `FileInputStream` ([§22.4.3](#)) or `FileOutputStream` ([§22.16.3](#)) to contain it.

```
public final class FileDescriptor {  
    public static final FileDescriptor in = ...;  
    public static final FileDescriptor out = ...;  
    public static final FileDescriptor err = ...;  
    public boolean valid();  
}
```

22.26.1 `public static final FileDescriptor in = ...`

A file descriptor for the standard input stream. Usually, this file descriptor is not used directly, but rather the input stream known as `System.in` ([§20.18.1](#)).

22.26.2 `public static final FileDescriptor out = ...`

A file descriptor for the standard output stream. Usually, this file descriptor is not used directly, but rather the output stream known as `System.out` ([§20.18.2](#)).

22.26.3 `public static final FileDescriptor err = ...`

A file descriptor for the standard error output stream. Usually, this file descriptor is not used directly, but rather the output stream known as `System.err` ([§20.18.3](#)).

22.26.4 `public boolean valid()`

If this `FileDescriptor` is valid (represents an active connection to a file or other active I/O connection), then the result is `true`. Otherwise, the result is `false`.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.27 The Class java.io.IOException

The class `IOException` is the general class of exceptions produced by failed or interrupted input/output operations. Subclasses of `IOException` include:

```
EOFException
FileNotFoundException
InterruptedIOException
UTFDataFormatException
public class IOException extends Exception {
    public IOException();
    public IOException(String s);
}
```

22.27.1 `public IOException()`

This constructor initializes a newly created `IOException` with `null` as its error message string.

22.27.2 `public IOException(String s)`

This constructor initializes a newly created `IOException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.28 The Class java.io.EOFException

An `EOFException` is thrown to indicate that an input operation has encountered end of file. Note that some Java input operations react to end of file by returning a distinguished value (such as `-1`) rather than by throwing an exception.

```
public class EOFException extends IOException {  
    public EOFException();  
    public EOFException(String s);  
}
```

22.28.1 `public EOFException()`

This constructor initializes a newly created `EOFException` with `null` as its error message string.

22.28.2 `public EOFException(String s)`

This constructor initializes a newly created `EOFException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.29 The Class java.io.FileNotFoundException

A `FileNotFoundException` is thrown to indicate that no actual file could be opened for a specified path name. See constructors `FileInputStream` ([§22.4.1](#), [§22.4.2](#)) and `FileOutputStream` ([§22.16.1](#), [§22.16.2](#)).

```
public class FileNotFoundException extends IOException {  
    public FileNotFoundException();  
    public FileNotFoundException(String s);  
}
```

22.29.1 `public FileNotFoundException()`

This constructor initializes a newly created `FileNotFoundException` with `null` as its error message string.

22.29.2 `public FileNotFoundException(String s)`

This constructor initializes a newly created `FileNotFoundException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


22.30 The Class java.io.InterruptedIOException

An `InterruptedIOException` is thrown to indicate that an input or output transfer has been terminated because the thread performing it was interrupted. The field `bytesTransferred` indicates how many bytes were successfully transferred before the interruption occurred.

```
public class InterruptedIOException extends IOException {  
    public int bytesTransferred = 0;  
    public InterruptedIOException();  
    public InterruptedIOException(String s);  
}
```

22.30.1 `public int bytesTransferred = 0;`

The number of bytes that had been transferred by the I/O operation before the operation was interrupted.

22.30.2 `public InterruptedIOException()`

This constructor initializes a newly created `InterruptedIOException` with `null` as its error message string.

22.30.3 `public InterruptedIOException(String s)`

This constructor initializes a newly created `InterruptedIOException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

22.31 The Class java.io.UTFDataFormatException

A `UTFDataFormatException` is thrown to indicate that a problem occurred in converting data from Java modified UTF-8 format. See method `readUTF` of `DataInput` ([§22.1.15](#)).

```
public class UTFDataFormatException extends IOException {  
    public UTFDataFormatException();  
    public UTFDataFormatException(String s);  
}
```

22.31.1 `public UTFDataFormatException()`

This constructor initializes a newly created `UTFDataFormatException` with `null` as its error message string.

22.31.2 `public UTFDataFormatException(String s)`

This constructor initializes a newly created `UTFDataFormatException` by saving a reference to the error message string `s` for later retrieval by the `getMessage` method ([§20.22.3](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Package com.ms.com

Classes in the package com.ms.com are necessary when using COM services from Java, or when implementing COM services using Java.

Note that only “trusted” Java applets can use COM services. Trusted applets are those that are loaded from the local machine or are packaged in a cabinet (.CAB) file that has a digital signature.

Classes

[ComContext](#)

[LicenseMgr](#)

[SafeArray](#)

[Variant](#)

Interfaces

[ILicenseMgr](#)

Exceptions

[ComException](#)

[ComFailException](#)

[ComSuccessException](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class ComContext

```
final public class ComContext
{
    // Fields
    public static final int INPROC_SERVER;
    public static final int INPROC_HANDLER;
    public static final int LOCAL_SERVER;
    public static final int INPROC_SERVER16;
    public static final int REMOTE_SERVER;
    public static final int INPROC_HANDLER16;
    public static final int INPROC_SERVERX86;
    public static final int INPROC_HANDLERX86;
}
```

The **ComContext** class defines constants for use with the **ILicenseMgr** interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ComContext.XXXX

```
public static final int INPROC_SERVER = 0x01;
public static final int INPROC_HANDLER = 0x02;
public static final int LOCAL_SERVER = 0x04;
public static final int INPROC_SERVER16 = 0x08;
public static final int REMOTE_SERVER = 0x10;
public static final int INPROC_HANDLER16 = 0x20;
public static final int INPROC_SERVERX86 = 0x40;
public static final int INPROC_HANDLERX86 = 0x80;
```

Remarks

Each constant specifies a context in which a COM class is to be run.

Const	Meaning
INPROC_SERVER	server DLL
INPROC_HANDLER	handler DLL
LOCAL_SERVER	server EXE
INPROC_SERVER16	16-bit server
REMOTE_SERVER	remote
INPROC_HANDLER16	16-bit handler DLL
INPROC_SERVERX86	16-bit server
INPROC_HANDLERX86	16-bit handler DLL

caller)
INPR Wx86 server
OC_S DLL (runs in
ERVE same
RX86 process as
caller)
INPR Wx86
OC_H handler DLL
ANDL (runs in
ERX8 same
6 process as
caller)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class LicenseMgr

```
public class LicenseMgr
    implements ILicenseMgr
{
}
```

The **LicenseMgr** class implements the **ILicenseMgr** interface, which allows you to use licensed COM components or controls from your Java application.

The **LicenseMgr** class itself has no methods; you don't use it directly. Simple create an instance of **LicenseMgr** and then use the **ILicenseMgr** interface to manipulate it. For example:

```
ILicenseMgr licmgr = (ILicenseMgr)new LicenseMgr();
// use licmgr interface
```

See the [ILicenseMgr](#) interface for more information on how to use licensed COM components.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class SafeArray

```
public final class SafeArray
{
}
```

The **SafeArray** class is a Java wrapper for the SAFEARRAY data type defined by COM.

The **SafeArray** class itself has no methods; you typically don't need to manipulate its contents yourself. The most common use of **SafeArray** is as a bookmark. For example, when using Data Access Objects (DAO), you get a **SafeArray** object as a bookmark. The contents of the bookmark itself are not meaningful; you simply pass it back to another DAO method when you need to refer to a bookmarked location.

For example, to save your position within a **RecordSet** while using DAO:

```
SafeArray currRec = recordset.getBookmark();
// do other stuff
// come back later
recordset.setBookmark( currRec );
```

There is never a need to examine the value of the bookmark.

Various other properties in DAO are also bookmarks. For example, the **CacheStart** and **LastModified** properties in the **RecordSet** interface are also bookmarks, and are represented in Java using the **SafeArray** class.

For more information on DAO, see the DAO SDK.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class Variant

```
public final class com.ms.com.Variant
{
    // Fields
    public static final short VariantEmpty;
    public static final short VariantNull;
    public static final short VariantShort;
    public static final short VariantInt;
    public static final short VariantFloat;
    public static final short VariantDouble;
    public static final short VariantCurrency;
    public static final short VariantDate;
    public static final short VariantString;
    public static final short VariantDispatch;
    public static final short VariantError;
    public static final short VariantBoolean;
    public static final short VariantVariant;
    public static final short VariantObject;
    public static final short VariantByte;
    public static final short VariantTypeMask;
    public static final short VariantArray;
    public static final short VariantByref;
    // Constructors
    public          Variant ();
    // Methods
    public native void    changeType(short vartype);
    public Variant clone();
    public Variant cloneIndirect();
    protected void    finalize();
    public          short  getvt();
    public          void   getEmpty();
    public          void   getNull();
    public          short  getShort();
    public          int    getInt();
    public          float  getFloat();
    public          double getDouble();
    public          long   getCurrency();
    public          double getDate();
    public native String getString();
    public native Object getDispatch();
    public          int    getError();
    public native boolean getBoolean();
    public native Object getObject();
    public native byte   getByte();
    public native short  getShortRef();
    public native int    getIntRef();
    public native float  getFloatRef();
    public native double getDoubleRef();
    public native long   getCurrencyRef();
    public native double getDateRef();
    public native String getStringRef();
    public native Object getDispatchRef();
    public native int    getErrorRef();
    public native boolean getBooleanRef();
    public native Object getObjectRef();
    public native byte   getByteRef();
}
```



```

public void noParam();
public void putEmpty();
public void putNull();
public void putShort(short val);
public void putInt(int val);
public void putFloat(float val);
public void putDouble(double val);
public void putCurrency(long val);
public void putDate(double val);
public native void putString(String val);
public native void putDispatch(Object val);
public void putError(int val);
public void putBoolean(boolean val);
public native void putObject(Object val);
public void putByte(byte val);
public native void putShortRef(short val);
public native void putIntRef(int val);
public native void putFloatRef(float val);
public native void putDoubleRef(double val);
public native void putCurrencyRef(long val);
public native void putDateRef(double val);
public native void putStringRef(String val);
public native void putDispatchRef(Object val);
public native void putErrorRef(int val);
public native void putBooleanRef(boolean val);
public native void putObjectRef(Object val);
public native void putByteRef(byte val);
public native short toShort();
public native int toInt();
public native float toFloat();
public native double toDouble();
public native long toCurrency();
public native double toDate();
public native String toString();
public native Object toDispatch();
public native int toError();
public native boolean toBoolean();
public native Object toObject();
public native byte toByte();
public native void VariantClear();
}

```

The **Variant** class is the Java wrapper for the VARIANT structure defined by the Component Object Model (COM). A VARIANT structure can store any one of many different types of values, and is typically used to pass parameters to interfaces derived from **IDispatch**; a VARIANT can be used to pass a parameter either by value or by reference.

Use the Java **Variant** class when using COM services from Java, or when implementing COM services using Java.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.VariantXXXX

<code>public static final short VariantEmpty</code>	<code>= 0;</code>
<code>public static final short VariantNull</code>	<code>= 1;</code>
<code>public static final short VariantShort</code>	<code>= 2;</code>
<code>public static final short VariantInt</code>	<code>= 3;</code>
<code>public static final short VariantFloat</code>	<code>= 4;</code>
<code>public static final short VariantDouble</code>	<code>= 5;</code>
<code>public static final short VariantCurrency</code>	<code>= 6;</code>
<code>public static final short VariantDate</code>	<code>= 7;</code>
<code>public static final short VariantString</code>	<code>= 8;</code>
<code>public static final short VariantDispatch</code>	<code>= 9;</code>
<code>public static final short VariantError</code>	<code>= 10;</code>
<code>public static final short VariantBoolean</code>	<code>= 11;</code>
<code>public static final short VariantVariant</code>	<code>= 12;</code>
<code>public static final short VariantObject</code>	<code>= 13;</code>
<code>public static final short VariantByte</code>	<code>= 17;</code>
<code>public static final short VariantTypeMask</code>	<code>= 0xffff;</code>
<code>public static final short VariantArray</code>	<code>= 0x2000;</code>
<code>public static final short VariantByref</code>	<code>= 0x4000;</code>

Remarks

These constants describe the types of values that a **Variant** object can contain.

The **VariantByref** constant indicates that the value is being passed by reference, that is, any changes that the callee makes to the value of a parameter are visible to the caller.

The **VariantTypeMask** constant is useful if you don't care about the type of the value being passed, but are interested in the "shape," that is, whether it is a **VariantByref** or not.

You can use these contents when examining the return value of the **getvt** method. You can also pass one of these constants when calling the **changeType** method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.Variant

```
public Variant( );
```

Remarks

Constructs a new **Variant** object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.changeType

```
public native void changeType(short vartype);
```

Parameters

vartype The destination type. Must be one of the **Variant.VariantXXXX** constants.

Remarks

Converts the value stored in the **Variant** object to the specified type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.clone

```
public Variant clone( );
```

Remarks

Clones a **Variant** object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.cloneIndirect

```
public Variant cloneIndirect( );
```

Remarks

Creates a copy of the **Variant** object without the **VariantByref** flag set. See the Win32 API **VariantCopyInd** for more information.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.finalize

protected void finalize();

Remarks

Destroys a **Variant** object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.getvt

```
public short getvt( );
```

Remarks

Retrieves the type of the value currently stored in the **Variant**. The return value is some combination of the **Variant.VariantXXXX** constants.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.getXXXX

```
public      void    getEmpty();
public      void    getNull();
public      short   getShort();
public      int     getInt();
public      float   getFloat();
public      double  getDouble();
public      long    getCurrency();
public      double  getDate();
public native String getString();
public native Object getDispatch();
public      int     getError();
public native boolean getBoolean();
public native Object getObject();
public      byte    getByte();
```

Remarks

Retrieves the value stored in the **Variant** object. For values passed by reference, use the getXXXXRef methods.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.getXXXXRef

```
public native short  getShortRef();
public native int    getIntRef();
public native float  getFloatRef();
public native double getDoubleRef();
public native long   getCurrencyRef();
public native double getDateRef();
public native String getStringRef();
public native Object getDispatchRef();
public native int    getErrorRef();
public native boolean getBooleanRef();
public native Object getObjectRef();
public native byte   getByteRef();
```

Remarks

Retrieves the value stored in the **Variant** object after it was used to pass a parameter by reference.

In the case of using COM services from Java, when you use a **Variant** object to pass a value by reference, you must call the appropriate **putXXXXRef** method before passing the **Variant**. This is needed to allocate the space in which the value will be placed by the callee. Then call the appropriate **getXXXXRef** method to read the value; changes are not made to the variable you passed to the **putXXXXRef** method.

For example, suppose you are passing an integer to a method that takes an integer parameter by reference. You would use code like the following:

```
// IFoo is a COM interface with a ChangeInt method
// foo is a variable of type IFoo
Variant var = new Variant();
var.putIntRef(0);           // allocate the space needed
foo.ChangeInt(var);        // pass var to a COM method
int i = var.getIntRef();    // get the value back
```

This example calls **putIntRef** before passing the **Variant**, even if the `ChangeInt` method doesn't use the value it receives.

In the case of implementing a COM service using Java, you can use **getXXXXRef** to read the value passed in by the caller.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.noParam

```
public void noParam( );
```

Remarks

Makes the **Variant** object a placeholder for an omitted optional parameter. This is useful when using COM methods that have optional **Variant** parameters; these are indicated with the **optional** attribute in the ODL (Object Definition Language) description of the interface. When methods with optional parameters are exposed in Java, the Java version declares all the parameters.

When you wish to specify a value for an optional parameter, pass a regular **Variant** object when calling the method. When you wish to omit an optional parameter, simply create a **Variant** object, call the **noParam** method on it, and pass it in place of an actual value.

For example, with Data Access Objects (DAO), the interface **Workspace** defines a method named **OpenDatabase** which takes four parameters, three of which are optional. You could call **OpenDatabase** and specify only the first parameter as follows:

```
// myDB is a variable of type dao3032.Database
// myWorkspc is a variable of type dao3032.Workspace

Variant opt = new Variant();
opt.noParam();
myDB = myWorkspc.OpenDatabase("Inventory", opt, opt, opt);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.putXXXX

```
public      void    putEmpty();
public      void    putNull();
public      void    putShort(short  val);
public      void    putInt(int     val);
public      void    putFloat(float  val);
public      void    putDouble(double val);
public      void    putCurrency(long  val);
public      void    putDate(double  val);
public native void    putString(String val);
public native void    putDispatch(Object val);
public      void    putError(int     val);
public      void    putBoolean(boolean val);
public native void    putObject(Object val);
public      void    putByte(byte     val);
```

Parameters

val The value to be passed.

Remarks

Sets the value stored in the **Variant** object. For values passed by reference, use the [putXXXXRef](#) methods.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.putXXXXRef

```
public native void putShortRef(short val);
public native void putIntRef(int val);
public native void putFloatRef(float val);
public native void putDoubleRef(double val);
public native void putCurrencyRef(long val);
public native void putDateRef(double val);
public native void putStringRef(String val);
public native void putDispatchRef(Object val);
public native void putErrorRef(int val);
public native void putBooleanRef(boolean val);
public native void putObjectRef(Object val);
public native void putByteRef(byte val);
```

Parameters

val The value to be passed.

Remarks

Sets the value stored in the **Variant** object when it is used to pass a parameter by reference.

In the case of using COM services from Java, when you use a **Variant** object to pass a value by reference, you must call the appropriate **putXXXXRef** method before passing the **Variant**. This is needed to allocate the space in which the value will be placed by the callee. Then call the appropriate **getXXXXRef** method to read the value; changes are not made to the variable you passed to the **putXXXXRef** method.

For example, suppose you are passing an integer to a method that takes an integer parameter by reference. You would use code like the following:

```
// IFoo is a COM interface with a ChangeInt method
// foo is a variable of type IFoo
Variant var = new Variant();
var.putIntRef(0); // allocate the space needed
foo.ChangeInt(var); // pass var to a COM method
int i = var.getIntRef(); // get the value back
```

This example calls **putIntRef** before passing the **Variant**, even if the `ChangeInt` method doesn't use the value it receives.

In the case of implementing COM services from Java, you call **putXXXXRef** to specify the value that you return to the caller. You must use the correct type when setting the value; for example, if the **Variant** object contains a double value, you must call **putDoubleRef**.

Throws

A **ClassCastException** if you call a method that does not match the type of the variable currently

stored in the object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.toXXXX

```
public native short  toShort();
public native int    toInt();
public native float  toFloat();
public native double toDouble();
public native long   toCurrency();
public native double toDate();
public native String toString();
public native Object toDispatch();
public native int    toError();
public native boolean toBoolean();
public native Object toObject();
public native byte   toByte();
```

Remarks

Gets the value stored in the **Variant** object, cast to the specified type.

These methods do not modify the value stored within the **Variant** object.

Throws

A **ClassCastException** if the cast fails.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Variant.Clear

```
public native void VariantClear( );
```

Remarks

Clears a **Variant** object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Interface **ILicenseMgr**

```
public interface ILicenseMgr
{
    // Methods
    public Object createInstance(String clsid,
        Object punkOuter, ComContext clsctx) throws ComException;
    public Object createWithLic(String lic, String clsid,
        Object punkOuter, ComContext clsctx) throws ComException;
}
```

The **ILicenseMgr** interface allows you to use licensed COM components or controls from your Java application.

A licensed control cannot be instantiated except by legal users. A design-time license allows a developer to embed the control in his or her applications. When the developer distributes the application, a run-time license allows the control to be instantiated when the end-user runs the application.

To use **ILicenseMgr** with a licensed control, call the **createWithLic** method and specify the license string explicitly to create an instance of the control.

An example of a licensed component is the **DBEngine** component in Data Access Objects (DAO); you must use **ILicenseMgr** when using DAO.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ILicenseMgr.CreateInstance

public Object CreateInstance(String clsid, Object punkOuter, ComContext ctxFlags)
throws ComException;

Parameters

clsid The CLSID of the COM class you want to instantiate, specified in string form (for example, "{EE008642-64A8-11CE-920F-08002B369A33}").

punkOuter If the object is being created as part of an aggregate, this parameter specifies the aggregating object. Otherwise, you can simply pass **null** for this parameter.

ctxFlags A ComContext value representing the context in which the new object should be created.

Remarks

Creates an instance of the specified COM class.

Warning In general, you should not call this method. The appropriate means of creating an instance of a COM class is to use the **new** operator on the Java wrapper generated by the Java Type Library Wizard (or JavaTLB). This method should be called only in special situations, and only if you are familiar with low-level COM programming.

Throws

ComFailException if there was an error.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ILicenseMgr.createWithLic

public Object createWithLic(String lic, String clsid, Object punkOuter, ComContext ctxFlags)
throws ComException;

Parameters

lic The license for the COM class you want to instantiate.

clsid The CLSID of the COM class you want to instantiate, specified in string form (for example, "{00025E15-0000-0000-C000-000000000046}").

punkOuter If the object is being created as part of an aggregate, this parameter specifies the aggregating object. Otherwise, you can simply pass **null** for this parameter.

ctxFlags A ComContext value representing the context in which the new object should be created..

Remarks

Creates a new instance of the specified licensed COM class.

Throws

ComFailException if there was an error.

Example

The following is a helper class with single method, `create`, that wraps a call to **CreateWithLic** to create an instance of the DAO (Data Access Objects) engine. By using this method, you can create an instance of a specific licensed component without having to specify the license or the CLSID every time.

```
import dao3032.*;
import com.ms.com.*;

public class dao_dbengine
{
    static public _DBEngine create()
    {
        _DBEngine result;
        ILicenseMgr mgr = new LicenseMgr();

        result = (_DBEngine) mgr.createWithLic(
            "mjgqcqcejfchcijecpdhckcdjqigdejfccjri", // BSTR lic
            "{00025E15-0000-0000-C000-000000000046}", // BSTR clsid
            null, //
            IUnknown* punkOuter
                ComContext.INPROC_SERVER // ComContext
            ctxFlags
        );

        return result;
    }
}
```

This example is found in the DAOSample sample application. The associated .CLASS file is also in the \REDIST directory.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class ComException

```
public abstract class com.ms.com.ComException
    extends RuntimeException
{
    // Constructors
    public ComException();
    public ComException(int hr);
    public ComException(String message);
    public ComException(int hr, String message);
    // Methods
    public int getHResult();
}
```

ComException is the Java class that wraps an HRESULT, the return type for most methods in the Component Object Model (COM). When calling COM methods from Java, you catch **ComException** objects to handle error conditions. Note that **ComException** is derived from **RuntimeException**, so the compiler does not check whether methods throw **ComException** objects or not; it is up to your discretion as to when you should use **try-catch** blocks.

When implementing COM classes using Java, you throw **ComException** objects to signal error conditions. Since **ComException** is an abstract class, you cannot construct a **ComException** object directly. Instead you construct either a **ComFailException** or a **ComSuccessException** object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ComException.ComException

```
public ComException( );  
public ComException( int hr );  
public ComException( String message );  
public ComException( int hr, String message );
```

Parameters

hr The HRESULT to return.

message The detail message.

Remarks

Constructs a new **ComException** object. Because **ComException** is an abstract class, this constructor is called only by the constructors of the classes derived from **ComException**.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ComException.getHResult

```
public int getHResult( );
```

Remarks

Retrieves the HRESULT returned by the COM method. Call this method when catching a **ComException** thrown by a COM method.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class ComFailException

```
public class com.ms.com.ComFailException
    extends com.ms.com.ComException
{
    // Constructors
    public ComFailException();
    public ComFailException(int hr);
    public ComFailException(String message);
    public ComFailException(int hr, String message);
}
```

ComFailException indicates a failure. The exception object wraps an HRESULT, the return type for most methods in the Component Object Model (COM). The default value of the HRESULT stored in a **ComFailException** is E_FAIL (0x80004005L). All failure HRESULTs have their most significant bit set.

The values of system-defined HRESULTs can be found in WINERROR.H, included with the Win32 SDK. For component-specific errors, see the documentation associated with the component.

For system-defined errors, the detail message stored in the **ComException** object is the string returned by the Win32 API function **FormatMessage**, which returns a brief message associated with the error. This detail message can be retrieved by calling getMessage (defined by the Throwable class).

The following table lists the values of some common HRESULTs:

Error	Value
E_UNEXPECTED (Unexpected failure)	0x80000000
E_NOTIMPL (Not implemented)	0x80000001
E_OUTOFMEMORY (Ran out of memory)	0x80000002
E_INVALIDARGUMENT (One or more arguments are invalid)	0x80000003
E_NOINTERFACE (No such interface supported)	0x80000004
E_POINTER (Invalid pointer)	0x80000005
E_HANDLE	0x80000006

(Invalid handle)	70006
E_ABORT (Operation aborted)	0x800 04004
E_FAIL (Unspecified error)	0x800 04005
E_ACCESSDENIED (General access denied error)	0x800 70005
E_NOTIMPL (Not implemented)	0x800 00001
DISP_E_UNKNOWNINTERFACE (Unknown interface)	0x800 20001
DISP_E_MBERNOTFOUND (Member not found)	0x800 20003
DISP_E_PARAMETERNOTFOUND (Parameter not found)	0x800 20004
DISP_E_TYPEMISMATCH (Type mismatch)	0x800 20005
DISP_E_UNKNOWNNAME (Unknown name)	0x800 20006
DISP_E_NO_NAMED_ARGUMENTS (No named arguments)	0x800 20007
DISP_E_BADVARTYPE (Bad variable type)	0x800 20008
DISP_E_EXCEPTION (Exception)	0x800 20009

occurred)

DISP_E_OV	0x800
ERFLOW	2000
(Out of	A
present	
range)	
DISP_E_BA	0x800
DINDEX	2000
(Invalid	B
index)	
DISP_E_UN	0x800
KNOWNLCI	2000
D (Memory is	C
locked)	
DISP_E_AR	0x800
RAYISLOCK	2000
ED (Memory	D
is locked)	
DISP_E_BA	0x800
DPARAMCO	2000
UNT (Invalid	E
number of	
parameters)	
DISP_E_PA	0x800
RAMNOTOP	2000
TIONAL	F
(Parameter	
not optional)	
DISP_E_BA	0x800
DCALLEE	20010
(Invalid	
callee)	
DISP_E_NO	0x800
TACCOLLECT	20011
ION (Does	
not support a	
collection)	

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComFailException.ComFailException

```
public ComFailException( );  
public ComFailException( int hr );  
public ComFailException( String message );  
public ComFailException( int hr, String message );
```

Parameters

hr The HRESULT to return.

message The detail message.

Remarks

Constructs a new **ComFailException** object. Throw a **ComFailException** object in order to signal an error condition when implementing a COM class using Java.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class ComSuccessException

```
public class com.ms.com.ComSuccessException
    extends com.ms.com.ComException
{
    // Constructors
    public ComSuccessException();
    public ComSuccessException(int hr);
    public ComSuccessException(String message);
    public ComSuccessException(int hr, String message);
}
```

ComSuccessException indicates a success. The exception object wraps an HRESULT, the return type for most methods in the Component Object Model (COM). The default value of the HRESULT stored in a **ComSuccessException** is S_FALSE (0x00000001L), which is used to indicate the successful completion of a method returning a Boolean FALSE value. Other success values are possible; all success HRESULTs have their most significant bit cleared.

To return an S_OK value when implementing a COM method in Java, simply have the method return as usual.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ComSuccessException.ComSuccessException

```
public ComSuccessException( );  
public ComSuccessException( int hr );  
public ComSuccessException( String message );  
public ComSuccessException( int hr, String message );
```

Parameters

hr The HRESULT to return.

message The detail message.

Remarks

Constructs a new **ComSuccessException** object. Throw a **ComSuccessException** object in order to signal a successful return when implementing a COM class using Java.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Overview: README for Microsoft Visual J++ Version 1.0

© 1996 Microsoft Corporation

This document includes updated information for the documentation provided with Microsoft® Visual J++™ Version 1.0. The information in this document is more up-to-date than that in the Help system. Many of the issues outlined in this document will be corrected in upcoming releases.

Please note that there may be last minute additions to the VJREAD.WRI file on the Visual J++ 1.0 CD that do not appear in this section of the online documentation. Please be sure to read the contents of the VJREAD.WRI file located in the \MSDEV directory on your Visual J++ 1.0 CD. You can access VJREAD.WRI by opening the Release Notes item in the Microsoft Visual J++ program group.

Select any topic below for more information:

- [Setup](#)
- [Development Environment](#)
- [Using Visual J++ with Windows 95: Batch Files](#)
- [Compiler and Tools](#)
- [Miscellaneous](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setup

- [Setup Troubleshooting Information](#)
- [Windows 95](#)
- [DLL Files Hidden on Windows 95](#)
- [Installing Updated Visual C++, Visual C++ for Macintosh, or Fortran Powerstation Packages](#)
- [Visual C++ Standard Edition not Compatible with Visual J++](#)
- [Installing/Uninstalling on a Dual-Boot Machine](#)
- [Updating System DLLs](#)
- [Application Error During Online Registration](#)
- [Setup Cannot Write to the Registry](#)
- [Access Permission for Installation Directory](#)
- [Real-Mode CD-ROM Drivers Under Windows 95 May Cause Error During Setup](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setup Troubleshooting Information

Information regarding how to troubleshoot installation problems encountered during setup of Visual J++ 1.0 is available in the PSS.HLP file located in the \MSDEV\HELP directory on the CD-ROM. Open this file and click the Contents tab. Next, double-click Your First Stop For Visual J++ 1.0 Support, and then double-click Setup Troubleshooting Guide. This help topic will lead you to the appropriate information.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Windows 95

On some video adapters, when running setup under Windows 95, the setup progress bar displays improperly and mouse support is lost. To resolve this problem you will need to run a utility supplied by your display adapter manufacturer that adjusts the refresh rates. For more information please see the Windows 95 README file.

DLL Files Hidden on Windows 95

By default, the Windows 95 Explorer sets DLLs as a hidden file type. If you want to see DLLs listed in File Open dialogs you can do one of two things:

- Remove DLLs from the Windows 95 Explorer hidden files list.
- After running Visual J++ Setup, log off and then log on again. Setup will reset the DLL file type so that it is no longer hidden.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Installing Updated Visual C++, Visual C++ for Macintosh, or Fortran Powerstation Packages

If you install any of the above packages **after** installing Visual J++ version 1.0, you may be unable to run the packages correctly in Microsoft Developer Studio until you make an adjustment. Developer Studio will notify you of this situation when you first try to run the newly installed product. A message box will tell you that Developer Studio has detected an incompatible version of the product you are trying to run.

The incompatibility does not preclude you from using Visual J++. Simply click OK in the message box and you can then use Visual J++ 1.0. The incompatibility does, however, preclude you from running any of the other listed products until you make the needed adjustment.

The fix needed for this incompatibility is stored on the Visual J++ 1.0 CD. To fix the problem, re-run the Visual J++ 1.0 Setup program, choose Custom installation and uncheck all the options. This step will automatically update the necessary files on your disk. After doing so, you will be able to run your add-on language program in Microsoft Developer Studio.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Visual C++ Standard Edition Not Compatible with Visual J++

Visual C++ Standard Edition does not support multiple packages. Therefore, you must install Visual J++ 1.0 and Visual C++ Standard Edition in separate directories.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Installing/Uninstalling on a Dual-Boot Machine

To install Visual J++ 1.0 on a dual-boot machine, run Setup once for each operating system. For example, boot the machine into Windows NT (version 4.0 or later) and run Setup. This will copy all the required files into the installation root directory (usually C:\MSDEV) and into the Windows NT \SYSTEM32 subdirectory. It will also set up a program group for Visual J++ and make the required registry entries. Next, reboot the machine into Windows 95 and run Setup again. You can choose the same installation directory. This will save disk space and installation time since Setup copies only those files that need replacement or updating. As a result, only system files will be copied to the system directory. Setup will also create a program group for Visual J++ and make the required registry entries for Windows 95. When you have finished the installations, you can use Visual J++ with either operating system. Because the operating systems do not share the same registry, Visual J++ will have to be configured separately for each one.

Note If you wish to uninstall Visual J++ from a dual-boot system, you will need to run the uninstall application from the operating system on which Visual J++ was originally installed. Therefore, if you originally installed Visual J++ using Windows 95, you will need to run the uninstall application from Windows 95. The Visual J++ registry entries will not be removed on the other operating system and will need to be removed manually.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Updating System DLLs

When the setup program updates system DLLs, it only updates older DLLs that are already installed. However, it updates the .SYM/.PDB files even if the DLL on the system is newer. This results in mismatched .SYM/.PDB and DLLs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Application Error During Online Registration

If you choose Online Registration after setting up Visual J++, you may get an application error in RegWiz.EXE. You can close the application error dialog box and restart your computer. After restarting your computer, you should be able to register Visual J++ electronically by clicking the "Register Visual J++" icon. If you are unable to run RegWiz, then you can fill out the registration card included in the Visual J++ package.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Setup Cannot Write to the Registry

Near the end of the Visual J++ installation, you may encounter a dialog box warning that "Setup Cannot Write To The Registry." The only option available is to click OK, at which time the setup continues. You can safely ignore this warning.

This warning is generated when installing Visual J++ from an account which does not have administrative privileges. This is a problem with the setup program only—it is not required that you have administrative privileges to install or use Visual J++.

The information Visual J++ stores in the registry is not critical. It is primarily information about the last few project and source files that were opened, window sizes/positions, and so on. If a registry entry is not present when Developer Studio is started, a new one is created with the original default settings.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Access Permission for Installation Directory

The Setup program requires that you have access permission for the root directory of the drive where you are installing Visual J++. Setup will fail if you do not have access rights. Under these circumstances, your system administrator must give you temporary access rights to the root directory for you to complete setup.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Real-Mode CD-ROM Drivers Under Windows 95 May Cause Error During Setup

If you are running Windows 95 and are using real-mode CD-ROM drivers, you will not be able to access long filenames from the Visual J++ CD. This may cause errors during Setup.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Development Environment

- [BRIEF Editor Emulation](#)
- [Project Dependencies May Affect Visual J++ Beta Projects](#)
- [Using Your Own Help Files for F1 Help in a Source File](#)
- [Using Decimal Numbers When Debugging in Hexadecimal Mode](#)
- [Manually Refreshing Source-Code Control Status](#)
- [Information Title Not Found Error](#)
- [Source-Code Control Integration with Visual J++ 1.0](#)
- [Visual SourceSafe: Error Accessing Source Code Control System](#)
- [The GIF and JPEG Files Supported by the Image Editor Cannot Be Loaded as Resources](#)
- [Bilevel GIF File Format Not Supported](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BRIEF Editor Emulation

The BRIEF system of window controls has not been completely emulated. The Create Window command brings up splitter bars within the active document window or activates movement of existing splitters. When the splitters are activated, ARROW keys allow placement of the splitters. Activated splitters are anchored by pressing ENTER. Change Window commands move between split panes. The Delete Window commands delete either horizontally or vertically adjacent panes.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


New Project Dependency Updates May Affect Projects Created With Visual J++ Beta Releases

Projects originally created with beta or trial versions of Visual J++ may receive the following error message when rebuilt with the final release version of Visual J++ 1.0:

- “One or more dependencies are out of date. Do you wish to rebuild them?”

This error message may appear every time you build.

If you encounter this error and it persists, use the following steps to correct this behavior:

1. Manually delete all files with the extension .DEP from your project.
2. Select Update All Dependencies from the Build menu. (Perform this step for both the Release and Debug versions of your project.)
3. Select Rebuild All from the Build menu. (Perform this step for both the Release and Debug versions of your project.)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Your Own Help Files for F1 Help in a Source File

You can customize Microsoft Developer Studio to get F1 help on keywords documented in a private or third-party help file. Once this feature is set up, Developer Studio adds a command to the Help menu allowing you to toggle the feature on or off. When the command is active, all source-file F1 help requests are passed to a specified external file or files rather than to Visual J++ Books Online. WinHelp 4.0 displays the topic. See Microsoft Developer Extension Help (EXTHELP.HLP) in the product \HELP directory for details on setting up Extension Help.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Decimal Numbers When Debugging in Hexadecimal Mode

When using the debugger in hexadecimal mode, it is possible to enter decimal numbers using the prefix `0n` (zero-n). For example:

```
0n1000
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Manually Refreshing Source-Code Control Status

If you have installed a source-code control system that conforms to the Microsoft Source Code Control Interface, under certain circumstances you may need to manually refresh the status of your files under source-code control. Microsoft Developer Studio automatically refreshes the source-code control status for files in a project workspace until the number of files reaches a certain threshold value. This value is the Project Threshold, set in the Registry under Source Control in the Microsoft Developer entry for the current user. After that value is reached, Developer Studio disables the automatic refresh for performance reasons. If you need to refresh the source-code control status of your files after the automatic refresh has been disabled, from the Tools menu, choose Source Control, and from the cascading menu, choose Refresh Status.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Information Title Not Found Error

If you install Microsoft Developer Studio into a directory other than the default, and receive a dialog titled "Information Title Not Found", verify that the path provided is correct and that it ends with a backslash. Click the Open button to continue after completing any necessary changes to the path. This problem is known to occur when certain double-byte characters with the trailing byte of 0x5c appear as the last character of the help directory name. This problem only occurs on operating systems that support double-byte characters such as Japanese Windows 95.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Source-Code Control Integration with Visual J++ 1.0

Visual J++ 1.0 does not ship with a source-code control system.

Microsoft Developer Studio provides facilities for integrating a source-code control system into the development environment. If you install a source-code control system that conforms to the Microsoft Common Source Code Control Interface, you can directly access source-code control functionality from the Developer Studio menus.

Visual Source Safe 4.0 conforms to the Microsoft Common Source Code Control Interface. Earlier versions of Source Safe do not. If you currently use a source-code control system other than Visual Source Safe, you may wish to contact your source-code control vendor to inquire about an update that conforms to the Microsoft Common Source Code Control Interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Visual SourceSafe: Error Accessing Source Code Control System

Using Microsoft Visual SourceSafe, you may encounter problems attempting to add more than 300 files to a project via Developer Studio's "Add To Source Control" menu item. Similar problems may occur when attempting to add files to a project already containing more than 300 files (including dependencies).

Symptoms include error messages such as the following:

Error accessing Source Code Control system. Check Installation.

Suggested work-arounds:

1. Add the files to the SourceSafe project directly from the SourceSafe Explorer.
2. Use RegEdit to edit the HKEY_CURRENT_USER\SOFTWARE\Microsoft\Developer Source Control key, and change the "ProjectThreshold" value to a value larger than the number of files in your project (including dependencies).

Note Increasing this value will impact overall Source Code Control performance. Only increase this value as much as necessary.

3. Contact Product Support Services for Microsoft Visual SourceSafe. See the information that came with your Microsoft SourceSafe product for details.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

The GIF and JPEG Files Supported by the Image Editor Cannot Be Loaded as Resources

The image editor has the capability to read and write GIF and JPEG graphic files. The Resource View only supports BMP files. GIF and JPEG files cannot be loaded in Resource View. Additionally, you cannot use the Insert Resource command to create GIF and JPEG files. To add a GIF or JPEG file to a project, you must use the Insert File command.

{ewl msdncd, EWGraphic, rea22a 0 /a "build.bmp"} To create a new GIF or JPEG file and add it to a project

- 1** From the File menu, choose New.
The New dialog box appears.
- 2** Select Bitmap File, and then choose OK.
The image editor appears.
- 3** Use the image editor to create the image.
- 4** From the File menu, choose Save As.
The Save As dialog box appears.
- 5** In the Save File As Type list box, select Bitmap File (*.bmp;*.dib;*.gif;*.jpg).
- 6** In the File Name box, append a GIF or JPG extension to the filename you choose.
- 7** Click the Save button.
- 8** From the Insert menu, choose Files Into Project.
The Insert Files Into Project dialog box appears.
- 9** In the List Files Of Type list box, choose Image Files (*.bmp;*.dib;*.gif;*.jpg).
- 10** Select the GIF or JPEG file you just saved.
- 11** Choose OK.

The GIF or JPEG file will appear in the File View. You can double-click the file icon and the image editor will appear with the image loaded.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Bilevel GIF File Format Not Supported

Bilevel (1-bit) GIF files may not display correctly when loaded into the image editor. Some other GIF file formats remain untested. If you encounter problems displaying or updating GIF files within the image editor, use another picture editor that supports a wider variety of GIF file formats to perform your updates.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using Visual J++ with Windows 95: Batch Files

Under Windows 95, Visual J++ is unable to stop a build that uses a batch file (.BAT) on the command line for an external project type. The Stop Build command on the Build menu will do nothing if a batch file is being run as part of the build. The build must continue to completion. You should only use NMAKE scripts (.MAK) to perform builds for external project types.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler and Tools

- Remote Connection Dialog Not Enabled
- The Debugger Only Supports Symbols Shorter than 255 Characters

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Remote Connection Dialog Not Enabled

Remote Connection, a utility allowing you to connect to other machines via the network or by serial modem cable, is not supported in this release.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


The Debugger Only Supports Symbols Shorter than 255 Characters

The debugger only supports symbol names that are shorter than 255 characters. Any symbol longer than the 255-character limit will be truncated, and it will not be evaluated and updated in the debugger.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Miscellaneous

- Win32 SDK Tool MIDL Ignores Intended Output File Case
- Long File Names
- Dynamic Creation of Redistributable ActiveX Controls Without License Key Will Fail

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Win32 SDK Tool MIDL Ignores Intended Output File Case

Some versions of the Win32 SDK tool MIDL will ignore the alphabetic case of an Interface Definition Language (.IDL) file when creating a Type Library (.TLB) file. This problem occurs if a .TLB file with the same name already exists in the target directory, but uses a different combination of upper- and lowercase characters. It is likely you will encounter this problem indirectly. For example, you may receive the error "Package <identifier> does not exist" when attempting to compile a Java program that imports the resulting type library.

As an example of this behavior, suppose a type library with the name TLIBRARY.tlb currently exists in your output directory. At some point, you decide a capitalization change is in order, so you change the Interface Definition Language file to Tlibrary.idl and recompile the file with MIDL. The problem occurs when MIDL reads the file Tlibrary.idl, detects the existence of TLIBRARY.tlb (case insensitive), and updates the new type library, preserving its file name as TLIBRARY.tlb.

To solve this problem, remove the type library from the intended output directory. MIDL will then create a new .TLB file, preserving the alphabetic case of input .IDL file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Long File Names

Microsoft Developer Studio will create long file names (LFN) when using AppWizard, ClassWizard, and some of the components in the Component Gallery. If you happen to be running on a system which does not correctly support LFN you must be aware of the system limitation and work around it.

For example, long file name support on Novell servers, 4.01 and earlier, is limited. This may interfere with project generation in Visual J++. In many instances you will not encounter problems. One known problem—because Novell shortens names to an 8.3 representation, it is possible to create projects with missing files. For example, creating a project called PROJECT1 will produce the following error,

```
PROJECT1,RC(69): Could not find the file res\PROJECT1Doc.ico
```

or

```
PROJECT1,RC(69): error RC2135 : file not found: res\PROJECT1Doc.ico
```

```
Error executing rc.exe.
```

This occurs because the PROJECT1Doc.ico file's name is shortened to PROJECT1.ico by the Netware driver, and later over-written by another file called PROJECT1.ico.

Also note that long file names with spaces are not viewable from a DOS session using the DIR command.

You can always work around these problems by specifying a project name of at most four characters on the New/Insert project dialog.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dynamic Creation of Redistributable ActiveX Controls Without License Key Will Fail

On machines that do not have Visual J++ installed, an attempt to dynamically create an instance of redistributable ActiveX controls supplied by Visual J++ will fail. However, creating an instance of the same control using a dialog box will be successful.

The redistributable controls supplied by Visual J++ are licensed. You will need to specify a valid license key when dynamically creating the control.

When Visual J++ is installed on a machine, that machine is licensed to use redistributable ActiveX controls. In this case, even if an instance of redistributable controls is dynamically created without a license key, the creation will still succeed. However, if this same application is executed on a machine without an installation of Visual J++ or some other product that distributes ActiveX controls (such as Microsoft Visual Basic), the dynamic creation process will fail if you do not specify a license key.

For more information on how to obtain a valid license key, see the Knowledge Base article, Q151804, "PRB: Dynamic Creation of Redistributable Control" on <http://www.microsoft.com/kb/>. This Web link is subject to change.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Copyright Page

The Java™ Application Programming Interface

**James Gosling
Frank Yellin
The Java Team**

© 1996 Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

All rights reserved.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The release described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Sun Microsystems, Inc. (SUN) hereby grants to you a fully-paid, nonexclusive, nontransferable, perpetual, worldwide limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this specification. This license allows and is limited to the creation and distribution of clean room implementations of this specification that (i) are complete implementations of this specification, (ii) pass all test suites relating to this specification that are available from SUN, (iii) do not derive from SUN source code or binary materials, and (iv) do not include any SUN binary materials without an appropriate and separate license from SUN.

Java and JavaScript are trademarks of Sun Microsystems, Inc. Sun, Sun Microsystems, Sun Microsystems Computer Corporation, the Sun logo, the Sun Microsystems Computer Corporation logo, Java and HotJava are trademarks or registered trademarks of Sun Microsystems, Inc. UNIX® is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. All other product names mentioned herein are the trademarks of their respective owners.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Preface

[A Bit of History](#)

[About the Java Series](#)

[Contributors to the API](#)

[About the Java Packages](#)

[References](#)

[Other Books in the Java Series](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

A Bit of History

Java is a general-purpose object-oriented programming language. Its syntax is similar to C and C++ but omits semantic features that make C and C++ complex, confusing, and insecure. Java was initially developed to address the problems of building software for small distributed systems to embed in consumer devices. As such it was designed for heterogeneous networks, multiple host architectures, and secure delivery. To meet these requirements, compiled Java code had to survive transport across networks, operate on any client, and assure the client that it was safe to run.

The popularization of the World Wide Web helped catapult these attributes of Java into the limelight. The Internet demonstrated how interesting, media-rich content could be made accessible in simple ways. Web browsers like Mosaic enabled millions of people to roam the Net and made Web surfing part of popular culture. At last there was a medium where what you saw and heard was essentially the same whether you were on a Mac, PC or UNIX machine, connected to a high-speed network or a modem.

But with popularity comes scrutiny and soon Web enthusiasts felt that the content supported by the Web's HTML document format was too limited. HTML extensions like forms only highlighted those limitations while making it clear that no browser could include all the features users wanted. Extensibility was the answer. At just this time the Java programming language found itself looking for another application.

Sun's HotJava browser was developed to showcase Java's interesting properties by making it possible to embed Java programs inside Web pages. These Java programs, known as *applets*, are transparently downloaded into the HotJava browser along with the HTML pages in which they appear. Before being accepted by the browser, applets are carefully checked to make sure they are safe. Like HTML pages, compiled Java programs are network- and platform-independent. Applets behave the same regardless of where they come from or what kind of machine they are being loaded into.

The Web community quickly noticed that Java was something new and important. With Java as the extension language, a Web browser could have limitless capabilities. Programmers could write an applet once and it would then run on any machine, anywhere. Visitors to Java-powered Web pages could use the content found there with confidence that nothing would damage their machine.

With applets as the initial focus, Java has demonstrated a new way to make use of the Internet to distribute software. This new paradigm goes beyond browsers. We believe it is an innovation with the potential to change the course of computing.

Tim Lindholm
Senior Staff Engineer
JavaSoft
April, 1996

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


About the Java Series

The Java series provides definitive reference documentation for Java programmers and end users. They are written by members of the Java team and published under the auspices of JavaSoft, a Sun Microsystems business. The World-Wide-Web allows Java documentation to be made available over the Internet, either by downloading or as hypertext. Nevertheless, the world-wide interest in Java led us to write these books.

To learn the latest about Java or download the latest Java release, visit our World Wide Web site at <http://java.sun.com>. For updated information about the Java Series, including sample code, errata, and previews of forthcoming books, visit <http://www.javasoft.com/books/Series>.

We would like to thank the Corporate and Professional Publishing Group at Addison-Wesley for their partnership in putting together the Series. Our editor Mike Hendrickson and his team have done a superb job of navigating us through the world of publishing. Within Sun, the support of James Gosling, Ruth Hennigar, and Bill Joy of Sun Microsystems ensured that this series would have the resources it needed to be successful. A personal note of thanks to my children Christopher and James for putting a positive spin on the many trips to my office during the development of the Series.

Lisa Friendly
Series Editor

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Contributors to the API

Designers of Classes and Interfaces

Tom Ball	Bill Joy
Lee Boynton	Tim Lindholm
Patrick Chan	Jonathan Payne
David Connelly	Sami Shaio
Pavani Diwanji	Doug Stein
Amy Fowler	Arthur van Hoff
James Gosling	Chris Warth
Jim Graham	Frank Yellin
Herb Jellinek	

Testers

Carla Schroer Vajah Srinivasan
Kevin Smith Headley Williamson

Layout and Supplemental Documentation

Lisa Friendly	Kathy Walrath	Doug Kramer
James Gosling	Annette Wagner	
Jonni Kanerva	Frank Yellin	
Guy Steele		

{ewl msdncd.dll, ewcright, /c"Microsoft"}

About the Java Packages

These two volumes describes the Java Application Programming Interface (API), a standard set of libraries for writing Java programs. The libraries evolved over several years of writing Java code to implement a variety of systems, ranging from consumer device networks to animated user interfaces to operating systems to compilers. In 1995 the libraries were reorganized to support Internet programming, and thus the Java API was created. Many people, both from inside and outside Sun, have been involved in the design of the API.

Although the API hasn't reached perfection yet, we believe it's useful and hope to make it a ubiquitous layer, available to all Internet applications.

Have fun.

Arthur van Hoff

Introduction

These books are a reference manual for Java application and applet programmers. To make full use of it you should be familiar with the Java programming language and its core concepts such as object orientation, garbage collection, and multithreading.

The extent of the API and the choice of functionality have been driven by several factors. First and foremost, the API should be simple and easy to use. Parts of the API, such as the support for multithreading, might introduce functionality that is new to you, but we think you'll find these new concepts simpler and easier to use than in most other programming environments.

The libraries in these books are the first generation of an API for writing Internet programs. A simple form of an Internet program is an *applet*-a small Java program that can be embedded in an HTML page.

The API has been designed with the Java language in mind. Important Java features such as object orientation, garbage collection, and multithreading played an important role in the API design. Instead of taking existing libraries and simply rewriting them in Java, we took the opportunity to design and implement the API making full use of the Java language.

For Release 1.0, we've tried to stay away from certain complex functionality, such as video and 3D, so that library implementations can be ported easily. We can include only functionality that is not proprietary and that is easily implemented on many platforms.

We expect to add to the API, but not to subtract from it or change its behavior. The API documented in this book will remain available to all Java programs through future releases.

If you have ideas about how the API could be improved or how to implement some of the missing functionality, we would like to hear from you. Please send your ideas and implementations to java@java.sun.com.

Using these API Books

Don't get overwhelmed by the multitude of classes documented in these two books. The structure of the Java language encourages the programmer to break up libraries into many classes, each describing a small part of the functionality. The class diagrams on the back cover are a good starting point for getting an impression of the relationships between classes. Editor: Is that where the class diagrams are going to be?

As you design and implement Java programs, you should write short test programs to verify your understanding of the classes. When in doubt, try it out!

The Java web site, <http://java.sun.com/>, contains many excellent and sometimes interactive explanations that can help you along. Another good source of information is the newsgroup `comp.lang.java`.

Package Overview

This overview describes each package in the Java API, starting with the most general-purpose package (`java.lang`) and ending with one of the most specialized packages (`java.applet`). Each package groups classes and interfaces that have similar functionality. The API contains the following packages:

- Volume I: Core Packages
 - `java.lang`: The Java Language Package
 - `java.io`: The Java I/O Package
 - `java.util`: Miscellaneous Java utilities and data structures
 - `java.net`: The Java Networking Package
- Volume II: Window Toolkit and Applets
 - `java.awt`: The Abstract Window Toolkit (AWT) Package
 - `java.awt.image`: The AWT Image Package
 - `java.awt.peer`: The AWT Peer Package
 - `java.applet`: The Java Applet Package

java.lang: The Java Language Package

The `java.lang` package provides the classes and interfaces that form the core of the Java language and Virtual Machine. For example, the classes `Object`, `String`, and `Thread`, for example, are used by almost every program and are closely intertwined with the Java language definition. Other `java.lang` classes define the exceptions and errors that the Java Virtual Machine can throw.

Another set of `java.lang` classes provide wrappers for primitive types. For example, the `Integer` class provides objects to contain `int` values.

Still other classes, such as `ClassLoader`, `Process`, `Runtime`, `SecurityManager`, and `System`, provide access to system resources. For other generally useful classes, see the `java.util` package.

The `java.lang` package is imported automatically into every Java program.

java.io: The Java I/O Package

The `java.io` package provides a set of input and output streams used to read and write data to files or other input and output sources. Java streams are byte oriented and the classes defined here can be chained to implement more sophisticated stream functionality.

java.util: The Java Utility Package

The `java.util` package contains a collection of utility classes and related interfaces. It includes classes that provide generic data structures (`Dictionary`, `Hashtable`, `Stack`, `Vector`), string manipulation (`StringTokenizer`), and calendar and date utilities (`Date`).

The java.util package also contains the Observer interface and Observable class, which allow objects to notify one another when they change.

java.net: The Java Networking Package

The java.net package contains networking classes and interfaces, including classes that represent a URL and a URL connection, classes that implement a socket connection, and a class that represents an Internet address.

java.awt: The Abstract Window Toolkit (AWT) Package:

The java.awt package provides the standard graphical user interface (GUI) elements such as buttons, lists, menus, and text areas. It also includes containers (such as windows and menu bars) and higher-level components (such as dialogs for opening and saving files). The AWT contains two more packages: java.awt.image and java.awt.peer.

java.awt.image: The AWT Image Package

The java.awt.image package contains classes and interfaces for performing sophisticated image processing. These classes and interfaces can be used by applications that need to create or manipulate images and colors.

java.awt.peer: The AWT Peer Package

The java.awt.peer package contains interfaces used to connect AWT components to their window system-specific implementations (such as Motif widgets).

Unless you're creating a window system-specific implementation of the AWT, you shouldn't need to use the interfaces in the java.awt.peer package.

java.applet: The Applet Package

The java.applet package contains classes and interfaces for creating applets, which are programs intended to be embedded into HTML pages or otherwise transported across a network.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


References

IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985. Available from Global Engineering Documents, 15 Inverness Way East, Englewood Colorado 80112-5704 USA, 303-792-2181 or 800-854-7179.

The Unicode Standard: Worldwide Character Encoding, Version 1.0, Volume 1 ISBN 0-201-56788-1 and Volume 2 ISBN 0-201-60845-6. Additional information about Unicode 1.1 may be found at <ftp://unicode.org>.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Other Books in the Java Series

Arnold, Ken and James Gosling. *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, 1996, ISBN 0-201-63455-4.

Campione, Mary and Kathy Walrath. *The Java Language Tutorial: Object-Oriented Programming for the Internet*. Addison-Wesley, Reading, Massachusetts, 1996, ISBN 0-201-63454-6.

Gosling, James, Bill Joy and Guy Steele. *The Java Language Specification*. Addison-Wesley, Reading, Massachusetts, 1996, ISBN 0-201-63451-1.

Lindholm, Tim, and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, Massachusetts, 1996, ISBN 0-201-63452-X.

For more information on the Java Series by Addison-Wesley, go to
<http://www.aw.com/cp/javaseries.html>

{ewl msdncd.dll, ewcright, /c"Microsoft"}

All Packages

Java API Documentation 1.0.2

Java Packages

Java interfaces and classes are grouped into *packages*. The following lists the java packages, from which you can access interfaces and classes.

java.lang

Package that contains essential Java classes, including numerics, strings, objects, compiler, runtime, security, and threads. This is the only package that is automatically imported into every Java program.

java.io

Package that provides classes to manage input and output streams to read data from and write data to files, strings, and other sources.

java.util

Package that contains miscellaneous utility classes, including generic data structures, bit sets, time, date, string manipulation, random number generation, system properties, notification, and enumeration of data structures.

java.net

Package that provides classes for network support, including URLs, TCP sockets, UDP sockets, IP addresses, and a binary-to-text converter.

java.awt

Package that provides an integrated set of classes to manage user interface components such as windows, dialog boxes, buttons, checkboxes, lists, menus, scrollbars, and text fields. (AWT = Abstract Window Toolkit)

java.awt.image

Package that provides classes for managing image data, including color models, cropping, color filtering, setting pixel values, and grabbing snapshots.

java.awt.peer

Package that connects AWT components to their platform-specific implementations (such as Motif widgets or Microsoft Windows controls).

java.applet

Package that enables the creation of applets through the Applet class. It also provides several interfaces that connect an applet to its document and to resources for playing audio.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.lang

Classes

- 1.1 Class Boolean
- 1.2 Class Character
- 1.3 Class Class
- 1.4 Class ClassLoader
- 1.5 Class Compiler
- 1.6 Class Double
- 1.7 Class Float
- 1.8 Class Integer
- 1.9 Class Long
- 1.10 Class Math
- 1.11 Class Number
- 1.12 Class Object
- 1.13 Class Process
- 1.14 Class Runtime
- 1.15 Class SecurityManager
- 1.16 Class String
- 1.17 Class StringBuffer
- 1.18 Class System
- 1.19 Class Thread
- 1.20 Class ThreadGroup
- 1.21 Class Throwable

Interfaces

- 1.22 Interface Cloneable
- 1.23 Interface Runnable

Exceptions

- 1.24 Class ArithmeticException
- 1.25 Class ArrayIndexOutOfBoundsException
- 1.26 Class ArrayStoreException
- 1.27 Class ClassCastException
- 1.28 Class ClassNotFoundException
- 1.29 Class CloneNotSupportedException
- 1.30 Class Exception
- 1.31 Class IllegalAccessException
- 1.32 Class IllegalArgumentException
- 1.33 Class IllegalMonitorStateException
- 1.34 Class IllegalThreadStateException
- 1.35 Class IndexOutOfBoundsException
- 1.36 Class InstantiationException

- 1.37 Class InterruptedException
- 1.38 Class NegativeArraySizeException
- 1.39 Class NoSuchMethodException
- 1.40 Class NullPointerException
- 1.41 Class NumberFormatException
- 1.42 Class RuntimeException
- 1.43 Class SecurityException
- 1.44 Class StringIndexOutOfBoundsException

Errors

- 1.45 Class AbstractMethodError
- 1.46 Class ClassCircularityError
- 1.47 Class ClassFormatError
- 1.48 Class Error
- 1.49 Class IllegalAccessException
- 1.50 Class IncompatibleClassChangeError
- 1.51 Class InstantiationError
- 1.52 Class InternalError
- 1.53 Class LinkageError
- 1.54 Class NoClassDefFoundError
- 1.55 Class NoSuchFieldError
- 1.56 Class NoSuchMethodError
- 1.57 Class OutOfMemoryError
- 1.58 Class StackOverflowError
- 1.59 Class ThreadDeath
- 1.60 Class UnknownError
- 1.61 Class UnsatisfiedLinkError
- 1.62 Class VerifyError
- 1.63 Class VirtualMachineError

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.1 Class Boolean

```
public final class java.lang.Boolean
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static Boolean FALSE; §1.1.1
    public final static Boolean TRUE; §1.1.2

    // Constructors
    public Boolean(boolean value); §1.1.3
    public Boolean(String s); §1.1.4

    // Methods
    public boolean booleanValue(); §1.1.5
    public boolean equals(Object obj); §1.1.6
    public static boolean getBoolean(String name); §1.1.7

    public int hashCode(); §1.1.8
    public String toString(); §1.1.9
    public static Boolean valueOf(String s); §1.1.10
}
```

This class wraps a value of the primitive type boolean in an object. An object of type Boolean contains a single field whose type is boolean.

In addition, this class provides a number of methods for converting a boolean to a String and a String to a boolean, as well as other constants and methods useful when dealing with a boolean.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Boolean.FALSE

```
public final static Boolean FALSE = new Boolean(true)
```

The Boolean object corresponding to the primitive value false.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Boolean.TRUE

```
public final static Boolean TRUE = new Boolean(false)
```

The Boolean object corresponding to the primitive value true.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Boolean.Boolean

public Boolean(boolean value)

Allocates a Boolean object representing the value argument.

Parameters:

value- the value of the boolean

public Boolean(String s)

Allocates a Boolean object representing the value true if the string argument is not null and is equal, ignoring case, to the string "true". Otherwise, allocates a Boolean object representing the value false.

Parameters:

s- the string to be converted to a boolean

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Boolean.booleanValue

public boolean booleanValue()

Returns:

the primitive boolean value of this object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Boolean.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a Boolean object that contains the same boolean value as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Boolean.getBoolean

public static boolean getBoolean(String name)

The result is true if the system property (I-§1.18.9) named by the argument exists and is equal, ignoring case¹, to the string "true".

Parameters:

name- the system property name

Returns:

the boolean value of the system property.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Boolean.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Boolean.toString

public String toString()

If this object contains the value true, a string equal to "true" is returned. Otherwise, a string equal to "false" is returned.

Returns:

a string representation of this object.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Boolean.valueOf

public static Boolean valueOf(String s)

A new Boolean object is constructed. This Boolean contains the value true if the string argument is not null and is equal, ignoring case, to the string "true".

Parameters:

s - a string

Returns:

The Boolean value represented by the string.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Footnotes

¹In Java 1.0, the string had to be equal to the string "true", where the comparison was case sensitive. Beginning with Java 1.1, the test is case insensitive.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.2 Class Character

```
public final class java.lang.Character
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static int MAX_RADIX;      §1.2.1
    public final static char MAX_VALUE;    §1.2.2
    public final static int MIN_RADIX;     §1.2.3
    public final static char MIN_VALUE;    §1.2.4

    // Constructors
    public Character(char value); §1.2.5

    // Methods
    public char charValue(); §1.2.6
    public static int digit(char ch, int radix); §1.2.7
    public boolean equals(Object obj); §1.2.8
    public static char forDigit(int digit, int radix); §1.2.9
    public int hashCode(); §1.2.10
    public static boolean isDefined(char ch); §1.2.11
    public static boolean isDigit(char ch); §1.2.12
    public static boolean isJavaLetter(char ch); §1.2.13
    public static boolean isJavaLetterOrDigit(char ch); §1.2.14
    public static boolean isLetter(char ch); §1.2.15
    public static boolean isLetterOrDigit(char ch); §1.2.16
    public static boolean isLowerCase(char ch); §1.2.17
    public static boolean isSpace(char ch); §1.2.18
    public static boolean isTitleCase(char ch); §1.2.19
    public static boolean isUpperCase(char ch); §1.2.20
    public static char toLowerCase(char ch); §1.2.21
    public String toString(); §1.2.22
    public static char toTitleCase(char ch); §1.2.23
    public static char toUpperCase(char ch); §1.2.24
}
```

This class wraps a value of the primitive type char in an object. An object of type Character contains a single field whose type is char.

In addition, this class provides a number of methods for determining the type of a character, and converting characters from uppercase to lowercase and vice versa.

Many of the methods of class Character are defined in terms of a "Unicode attribute table" that specifies a name for every defined Unicode code point. The table also includes other attributes, such as a decimal value, an uppercase equivalent, a lowercase equivalent, and/or a titlecase equivalent. The Unicode attribute table is available on the World Wide Web as the file:

<ftp://unicode.org/pub/MappingTables/UnicodeData1.1.5.txt>


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.MAX_RADIX

```
public final static int MAX_RADIX = 36
```

The constant value of this field is the largest value permitted for the radix argument in radix-conversion methods such as the digit (I-§1.2.7) method, the forDigit (I-§1.2.9) method, and the toString (I-§1.8.21) method of class Integer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.MAX_VALUE1

```
public final static char MAX_VALUE = '\uffff'
```

The constant value of this field is the largest value of type char.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.MIN_RADIX

```
public final static int MIN_RADIX = 2
```

The constant value of this field is the smallest value permitted for the radix argument in radix-conversion methods such as the digit (I-§1.2.7) method, the forDigit (I-§1.2.9) method, and the toString (I-§1.8.21) method of class Integer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.MIN_VALUE2

```
public final static char MIN_VALUE = '\u0000'
```

The constant value of this field is the smallest value of type char.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.Character

public Character(char value)

Constructs a Character object and initializes it so that it represents the primitive value argument.

Parameters:

value- value for the new Character object

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.charValue

public char charValue()

Returns:

The primitive char value represented by this object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Character.digit

public static int digit(char ch, int radix)

Returns the numeric value of the character ch in the specified radix.

If the radix is not in the range `MIN_RADIX` is less than or equal to radix is less than or equal to `MAX_RADIX` or if the ch is not a valid digit in the specified radix, -1 is returned. A character is a valid digit if any of the following is true:

- The method `isDigit` ([\(I-§1.2.12\)](#)) is true of the character and the Unicode decimal digit value of the character (or its single-character decomposition) is less than the specified radix. In this case the decimal digit value is returned.
- The character is one of the uppercase Latin letters 'A' through 'Z' and its code is less than `radix + 'A' - 10`. In this case, `ch - 'A' + 10` is returned.
- The character is one of the lowercase Latin letters 'a' through 'z' and its code is less than `radix + 'a' - 10`. In this case, `ch - 'a' + 10` is returned.

Parameters:

ch- the character to be converted

radix- the radix

Returns:

the numeric value represented by the character in the specified radix.

See Also:

`forDigit` ([\(I-§1.2.9\)](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a Character object that represents the same char value as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Character.forDigit

public static char forDigit(int digit, int radix)

Determines the character representation for a specific digit in the specified radix. If the value of radix is not a valid radix, or the value of digit is not a valid digit in the specified radix, the null character ("      ") is returned.

The radix argument is valid if it is greater than or equal to MIN_RADIX (I- 1.2.3) and less than or equal to MAX_RADIX (I- 1.2.1). The digit argument is valid if 0 is less than or equal to digit is less than radix.

If the digit is less than 10, then '0' + digit is returned. Otherwise, the value 'a' + digit - 10 is returned.

Parameters:

digit- the number to convert to a character

radix- the radix.

Returns:

the char representation of the specified digit in the specified radix.

See Also:

digit (I- 1.2.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.IsDefined³

public static boolean isDefined(char ch)

Determines if a character has a defined meaning in Unicode. A character is defined if at least one of the following is true:

- It has an entry in the Unicode attribute table.
- Its value is in the range '\u3040' is less than or equal to ch is less than or equal to '\u9FA5"
- Its value is in the range '\uF900' is less than or equal to ch is less than or equal to '\uFA2D"

Parameters:

ch- the character to be tested

Returns:

true if the character has a defined meaning in Unicode; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isDigit4

public static boolean isDigit(char ch)

Determines whether the specified character is a digit. A character is considered to be a digit if it is not in the range '\u2000' through '\uu2000' and its Unicode name contains the word "DIGIT".

Parameters:

ch- the character to be tested

Returns:

true if the character is a digit; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isJavaLetter5

public static boolean isJavaLetter(char ch)

Determines whether the specified character is a "Java" letter, that is, the character is permissible as the first character in an identifier in the Java language.

A character is considered to be a Java letter if it is a letter, the ASCII dollar sign character '\$', or the underscore character '_'.

Parameters:

ch- the character to be tested

Returns:

true if the character is a Java letter; false otherwise.

See Also:

isLetter ([I-§1.2.15](#))

isLetterOrDigit ([I-§1.2.16](#))

isJavaLetterOrDigit ([I-§1.2.14](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isJavaLetterOrDigit

public static boolean isJavaLetterOrDigit(char ch)

Determines whether the specified character is a "Java" letter or digit, that is, the character is permissible as a non-initial character in an identifier in the Java language.

A character is considered to be a Java letter or digit if it is a letter, a digit, the ASCII dollar sign character '\$', or the underscore character '_'.

Parameters:

ch – the character to be tested

Returns:

true if the character is a Java letter or digit; false otherwise.

See Also:

isLetter ([I-§1.2.15](#))

isLetterOrDigit ([I-§1.2.16](#))

isJavaLetter ([I-§1.2.13](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isLetter

public static boolean isLetter(char ch)

Determines whether the specified character is a letter.

Parameters:

ch- the character to be tested

Returns:

true if the character is a letter; false otherwise.

See Also:

isJavaLetter [\(I-§1.2.13\)](#)

isJavaLetterOrDigit [\(I-§1.2.14\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isLetterOrDigit7

public static boolean isLetterOrDigit(char ch)

Parameters:

ch– the character to be tested

Returns:

true if the character is a letter or digit; false otherwise.

See Also:

isDigit [\(I-§1.2.12\)](#)

isLetter [\(I-§1.2.15\)](#)

isJavaLetter [\(I-§1.2.13\)](#)

isJavaLetterOrDigit [\(I-§1.2.14\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isLowerCase

`public static boolean isLowerCase(char ch)`

Determines whether the specified character is a lowercase character.

A character is lowercase if it is not in the range '\u2000' through '\u2FFF', the Unicode attribute table does not specify a mapping to lowercase for the character, and at least one of the following is true:

- the attribute table specifies a mapping to uppercase for the character.
- the name for the character contains the words "SMALL LETTER."
- the name for the character contains the words "SMALL LIGATURE."

Parameters:

`ch`– the character to be tested

Returns:

true if the character is lowercase; false otherwise.

See Also:

`isUpperCase` [\(I-§1.2.20\)](#)

`isTitleCase` [\(I-§1.2.19\)](#)

`toLowerCase` [\(I-§1.2.21\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.isSpace

`public static boolean isSpace(char ch)`

Determines if a character is white space.

This method returns true for the following five characters only:

<code>'\t'</code>	<code>\u0009</code>	HORIZONTAL TABULATION
<code>'\n'</code>	<code>\u000A</code>	NEW LINE
<code>'\f'</code>	<code>\u000C</code>	FORM FEED
<code>'\r'</code>	<code>\u000D</code>	CARRIAGE RETURN
<code>' '</code>	<code>\u0020</code>	SPACE

Parameters:

`ch`– the character to be tested

Returns:

true if the character is ISO-LATIN-1 white space; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.IsTitleCase

`public static boolean IsTitleCase(char ch)`

Determines if the character is a titlecase character.

There are four [Unicode characters](#) whose printed representations look like pairs of Latin letters. For example, there is an uppercase letter that looks like "LJ" and the corresponding lowercase letter looks like "lj". A third form, which looks like "Lj" that is the appropriate form to use when rendering a word in lowercase with initial capitals, as for a book title.

These are the [Unicode characters](#) for which this [method](#) returns true:

- LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
- LATIN CAPITAL LETTER L WITH SMALL LETTER J
- LATIN CAPITAL LETTER N WITH SMALL LETTER J
- LATIN CAPITAL LETTER D WITH SMALL LETTER Z

Parameters:

`ch`– the character to be tested

Returns:

true if the character is titlecase; false otherwise.

See Also:

`isUpperCase` [\(I-§1.2.20\)](#)

`isLowerCase` [\(I-§1.2.17\)](#)

`toTitleCase` [\(I-§1.2.23\)](#).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Character.toUpperCase10

public static boolean isUpperCase(char ch)

Determines whether the specified character is an uppercase character.

A character is uppercase if it is not in the range '\u2000' through '\u2FFF', the Unicode attribute table does not specify a mapping to uppercase for the character, and at least one of the following is true:

- the attribute table specifies a mapping to lowercase for the character.
- the name for the character contains the words "CAPITAL LETTER."
- or the name for the character contains the words "CAPITAL LIGATURE."

Parameters:

ch– the character to be tested

Returns:

true if the character is uppercase; false otherwise.

See Also:

isLowerCase [\(I-§1.2.17\)](#)

isTitleCase [\(I-§1.2.19\)](#)

toUpperCase [\(I-§1.2.24\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.ToLowerCase¹¹

public static char toLowerCase(char ch)

The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned.

A character has a lowercase equivalent if a lowercase mapping is specified for the character in the Unicode attribute table.

Note that some Unicode characters in the range 'u2000' through 'u2FFF' have lowercase mappings; this method does map such characters to their lowercase equivalents even though the method isUpperCase (I-§1.2.20) does not return true for such characters.

Parameters:

ch- the character to be converted

Returns:

the lowercase equivalent of the character, if any; otherwise the character itself.

See Also:

isLowerCase (I-§1.2.17).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Character.toString

public String toString()

Converts this Character object to a string. The result is a string whose length is 1 and whose sole component is the primitive char value represented by this object.

Returns:

a string representation of this object.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.toTitleCase¹²

`public static char toTitleCase(char ch)`

Converts the character argument to titlecase. A character has a titlecase equivalent if a titlecase mapping is specified for the character in the Unicode attribute table.

Note that some Unicode characters in the range 'u2000' through 'u2FFF' have titlecase mappings; this method does map such characters to their titlecase equivalents even though the method `isTitleCase` (I-§1.2.19) does not return true for such characters.

Parameters:

`ch`— the character to be converted

Returns:

the titlecase equivalent of the character, if any; otherwise the character itself.

See Also:

`isTitleCase` (I-§1.2.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Character.toUpperCase¹³

`public static char toUpperCase(char ch)`

Converts the character argument to uppercase. A character has an uppercase equivalent if a titlecase mapping is specified for the character in the Unicode attribute table.

Note that some Unicode characters in the range 'u2000' through 'u2000FFF' have uppercase mappings; this method does map such characters to their titlecase equivalents even though the method `isLowerCase` (I-§1.2.17) does not return true for such characters.

Parameters:

`ch`– the character to be converted

Returns:

the uppercase equivalent of the character, if any; otherwise the character itself.

See Also:

`isUpperCase` (I-§1.2.20).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹This field is new in Java 1.1.

²This field is new in Java 1.1.

³This method is new in Java 1.1.

⁴In Version 1.0, this version returns true only for the ten ASCII digits '0' through '9'.

⁵This method is new in Java 1.1

⁶This method is new in Java 1.1

⁷This method is new in Java 1.1

⁸In Version 1.0, this version returns true only for lowercase characters in the range '\u0000' to '\u00FF'.

⁹This method is new in Java 1.1.

¹⁰In Java 1.0, this method returns true only for uppercase characters in the range '\u0000' to '\u00FF'.

¹¹In Java 1.0, this method only works on characters in the range '\u0000' to '\u00FF'. For characters outside this range, the method returns its argument unchanged.

¹²This method is new in Java 1.1

¹³In Java 1.0, this method only works on characters in the range '\u0000' to '\u00FF'. For characters outside this range, the method returns its argument unchanged.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.3 Class Class

```
public final class java.lang.Class
    extends java.lang.Object (l-§1.12)
{
    // Methods
    public static Class forName(String className); §1.3.1
    public ClassLoader getClassLoader(); §1.3.2
    public Class[] getInterfaces(); §1.3.3
    public String getName(); §1.3.4
    public Class getSuperclass(); §1.3.5
    public boolean isInterface(); §1.3.6
    public Object newInstance(); §1.3.7
    public String toString(); §1.3.8
}
```

Instances of the class Class represent classes and interfaces in a running Java application.

There is no **public** constructor for the class Class. Class objects are constructed automatically by the Java Virtual Machine as classes are loaded and/or by calls to the **defineClass** method (l-§1.4.2) in the class loader.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Class.forName

public static Class **forName**(String **className**)
throws **ClassNotFoundException**

Returns the Class object associated with the class with the given string name.

For example, the following code fragment returns the run-time Class descriptor for the class named java.lang.Thread:

```
Class t = Class.forName("java.lang.Thread")/
```

Parameters:

className— the fully qualified name of the desired class

Returns:

the Class descriptor for the class with the specified name.

Throws

ClassNotFoundException (I-§1.28)

If the class could not be found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.getClassLoader

public `ClassLoader` `getClassLoader()`

Determines the class loader ([I-§1.4](#)) for the class.

Returns:

The class loader that created the class or interface represented by this object, or null if the class was not created by a class loader.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.getInterfaces

public Class[] getInterfaces()

Determines the interfaces implemented by the class or interface represented by this object.

If this object represents a class, the return value is an array containing objects representing all interfaces implemented by the class. The order of the interface objects in the array corresponds to the order of the interface names in the implements clause of the declaration of the class represented by this object.

If this object represents an interface, the array contains objects representing all interfaces extended by the interface. The order of the interface objects in the array corresponds to the order of the interface names in the extends clause of the declaration of the interface represented by this object.

If the class or interface implements no interfaces, the method returns an array of length 0.

Returns:

an array of interfaces implemented by this class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.getName

public String getName()

Returns:

the fully qualified name of the class or interface represented by this object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.getSuperclass

public Class getSuperclass()

If this object represents any class other than the class Object, then the object that represents the superclass of that class is returned.

If this object is the one that represents the class Object or this object represents an interface, null is returned.

Returns:

the superclass of the class represented by this object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.isInterface

public boolean isInterface()

Returns:

true if this object represents an interface; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.newInstance

public Object newInstance()
throws InstantiationException, IllegalAccessException

Creates a new instance of a class.

Returns:

A newly allocated instance of the class represented by this object. This is done exactly as if by a new expression with an empty argument list.

Throws

InstantiationException (I-§1.36)

If an application tries to instantiate an abstract class or an interface, or if the instantiation fails for some other reason.

Throws

IllegalAccessException (I-§1.31)

If the class or initializer is not accessible.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Class.toString

public String toString()

Converts the object to a string. The string representation is the string "class" or "interface" followed by a space and then the fully qualified name of the class.

Returns:

a string representation of the class object.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.4 Class ClassLoader

```
public abstract class java.lang.ClassLoader
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    protected ClassLoader(); §1.4.1

    // Methods
    protected final Class §1.4.2
        defineClass(byte data[], int offset, int length);
    protected final Class findSystemClass(String name); §1.4.3
    protected abstract Class §1.4.4
        loadClass(String name, boolean resolve);
    protected final void resolveClass(Class c); §1.4.5
}
```

The class ClassLoader is an abstract class. Applications implement subclasses of ClassLoader in order to extend the manner in which the Java Virtual Machine dynamically loads classes.

Normally, the Java Virtual Machine loads classes from the local file system in a platform-dependent manner. For example, on UNIX systems, the Virtual Machine loads classes from the directory defined by the CLASSPATH environment variable.

However, some classes may not originate from a file; they may originate from other sources, such as the network, or they could be constructed by an application. The method **defineClass** (I-§1.4.2) converts an array of bytes into an instance of class Class (I-§1.3). Instances of this newly defined class can be created using the newInstance method in class Class (I-§1.3.7).

The methods and constructors of objects created by a class loader may reference other classes. To determine the class(es) referred to, the Java Virtual Machine calls the **loadClass** method (I-§1.4.4) of the class loader that originally created the class. If the Java Virtual Machine only needs to determine if the class exists and if it does exist to know its superclass, the resolve flag is set to false. However if an instance of the class is being created or any of its methods are being called, the class must also be resolved. In this case the resolve flag is set to true, and the **resolveClass** method (I-§1.4.5) should be called.

For example, an application could create a network class loader to download class files from a server. Sample code might look like:

```
ClassLoader loader = new NetworkClassLoader(host, port);
Object main = loader.loadClass("Main", true).newInstance();
....
```

The network class loader subclass must define the method **loadClass** to load a class from the network. Once it has downloaded the bytes that make up the class, it should use the method **defineClass** to create a class instance. A sample implementation might be the following:

```
class NetworkClassLoader {
    String host;
    int port;
```



```

Hashtable cache = new Hashtable();
private byte loadClassData(String name)[] {
    // load the class data from the connection
    ...
}

public synchronized Class loadClass(String name,
    boolean resolve) {
    Class c = cache.get(name);
    if (c == null) {
        byte data[] = loadClassData(name);
        c = defineClass(data, 0, data.length);
        cache.put(name, c);
    }
    if (resolve)
        resolveClass(c);
    return c;
}
}

```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ClassLoader.ClassLoader

protected `ClassLoader()`

Constructs a new class loader and initializes it.

If there is a security manager, its `checkCreateClassLoader` method (I-§1.15.8) is called. This may result in a security exception (I-§1.43).

Throws

`SecurityException` (I-§1.43)

If the current thread does not have permission to create a new class loader.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ClassLoader.defineClass

protected final Class

defineClass(byte data[], int offset, int length)

Converts an array of bytes into an instance of class Class.

Parameters:

data- the bytes that make up the Class

offset- the start offset of the Class data

length- the length of the Class data

Returns:

the class object which was created from the data.

Throws

ClassFormatError (I-§1.47)

If the data does not contain a valid class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ClassLoader.findSystemClass

```
protected final Class findSystemClass(String name)  
throws ClassNotFoundException
```

Finds the system class with the specified name, loading it if necessary.

A system class is a class loaded from the local file system in a platform-dependent way. It has no class loader.

Parameters:

`name`— the name of the system Class

Returns:

a system class with the given name.

Throws

NoClassDefFoundError (I-§1.54)

If the Class is not found.

Throws

ClassNotFoundException (I-§1.28)

If it could not find a definition for the class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ClassLoader.loadClass

protected abstract Class

loadClass(String name, boolean resolve)
throws ClassNotFoundException

Requests the class loader to load a class with the specified name. The loadClass method is called by the Java Virtual Machine when a class loaded by a class loader first references another class. Every subclass of class ClassLoader must define this method.

If the resolve flag is true, the method should call the resolveClass method on the resulting class object.

Class loaders should use a hash table or other cache to avoid defining classes with the same name multiple times.

Parameters:

name- the name of the desired Class

resolve- true if the class must be resolved

Returns:

the resulting class, or null if it was not found.

Throws

ClassNotFoundException (I-§1.28)

If the class loader cannot find a definition for the class.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

ClassLoader.resolveClass

protected final void resolveClass(Class c)

Resolves the class so that an instance of the class can be created, or so that one of its methods can be called. This method should be called by loadClass if the resolve flag is true.

Parameters:

c- the Class instance to be resolved

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.5 Class Compiler

```
public final class java.lang.Compiler
    extends java.lang.Object (I-§1.12)
{
    // Methods
    public static Object command(Object any); §1.5.1
    public static boolean compileClass(Class clazz); §1.5.2
    public static boolean compileClasses(String string); §1.5.3
    public static void disable(); §1.5.4
    public static void enable(); §1.5.5
}
```

The Compiler class is provided in support of Java-to-native-code compilers and related services.

When the Java Virtual Machine first starts, it determines if a system property (I-§1.18.9) `java.compiler` exists. If so, it is assumed to be the name of a library (whose exact location and type is platform-dependent); the `loadLibrary` method (I-§1.18.13) in class `System` is called to load that library. If this loading succeeds, the function named `java_lang_Compiler_start()` in that library is called.

If there is no compiler available, these methods do nothing.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler.command

public static Object command(Object any)

The Compiler should examine the argument type and its fields and perform some documented operation. No specific operations are required.

Parameters:

any- an argument

Returns:

a compiler-specific value, or null if no compiler is available.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Compiler.compileClass

public static boolean compileClass(Class clazz)

The indicated class is compiled.

Parameters:

clazz- a class

Returns:

true if the compilation succeeds; false if the compilation fails or no compiler is available.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Compiler.compileClasses

public static boolean compileClasses(String string)

Indicates that the application would like the third-party software to compile all classes whose name matches the indicated string.

Parameters:

string- the name of the classes to compile

Returns:

true if the compilation succeeds; false if the compilation fails or no compiler is available.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler.disable

```
public static void disable()
```

Causes the Compiler to cease operation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Compiler.enable

```
public static void enable()
```

Causes the Compiler to resume operation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.6 Class Double

```
public final class java.lang.Double
    extends java.lang.Number (l-§1.11)
{
    // Fields
    public final static double MAX_VALUE; §1.6.1
    public final static double MIN_VALUE; §1.6.2
    public final static double NaN; §1.6.3
    public final static double NEGATIVE_INFINITY; §1.6.4
    public final static double POSITIVE_INFINITY; §1.6.5

    // Constructors
    public Double(double value); §1.6.6
    public Double(String s); §1.6.7

    // Methods
    public static long doubleToLongBits(double value); §1.6.8
    public double doubleValue(); §1.6.9
    public boolean equals(Object obj); §1.6.10
    public float floatValue(); §1.6.11
    public int hashCode(); §1.6.12
    public int intValue(); §1.6.13
    public boolean isInfinite(); §1.6.14
    public static boolean isInfinite(double v); §1.6.15
    public boolean isNaN(); §1.6.16
    public static boolean isNaN(double v); §1.6.17
    public static double longBitsToDouble(long bits); §1.6.18
    public long longValue(); §1.6.19
    public String toString(); §1.6.20
    public static String toString(double d); §1.6.21
    public static Double valueOf(String s); §1.6.22
}
```

This class wraps a value of the primitive type double in an object. An object of type Double contains a single field whose type is double.

In addition, this class provides a number of methods for converting a double to a String and a String to a double, as well as other constants and methods useful when dealing with a double.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Double.MAX_VALUE

public final static double

MAX_VALUE = 1.79769313486231570e+308d

The largest positive value of type double.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.MIN_VALUE

public final static double

MIN_VALUE = 4.94065645841246544e-324

The smallest positive finite value of type double.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.NaN

```
public final static double NaN = 0.0 / 0.0
```

A Not-a-Number (NaN) value of type double.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Double.NEGATIVE_INFINITY

public final static double

NEGATIVE_INFINITY = -1.0 / 0.0

The negative infinity of type double.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.POSITIVE_INFINITY

```
public final static double
```

```
    POSITIVE_INFINITY = 1.0 / 0.0
```

The positive infinity of type double.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Double.Double

public Double(double value)

Constructs a newly allocated Double object that represents the primitive double argument.

Parameters:

value- the value to be represented by the Double

public Double(String s)
throws NumberFormatException

Constructs a newly allocated Double object that represents the floating-point value of type double represented by the string. The string is converted to a double value as if by the valueOf method (I-§1.6.22).

Parameters:

s- a string to be converted to a Double

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.doubleToLongBits

public static long doubleToLongBits(double value)

The result is a representation of the floating-point argument according to the IEEE 754 floating-point "double format" bit layout.

Bit 63 represents the sign of the floating-point number; bits 62-52 represent the exponent; bits 51-0 represent the significand (sometimes called the mantissa) of the floating-point number.

If the argument is positive infinity, the result is 0x7ff0000000000000L.

If the argument is negative infinity, the result is 0xfff0000000000000L.

If the argument is NaN, the result is 0x7ff8000000000000L.

Parameters:

value- a double precision floating point number

Returns:

the bits that represent the floating point number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.doubleValue

public double doubleValue()

Returns:

the double value represented by this object.

Overrides:

doubleValue in class Number (I-§1.11.1).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Double.equals

public boolean equals(Object obj)

The result is true if the argument is not null and is a Double object that represents a double that has the identical bit pattern to the bit pattern of the double represented by this object.

Note that in most cases, for two instances of class Double, d1 and d2, the value of d1.equals(d2) is true if d1.doubleValue() == d2.longValue() also has the value true. However, there are two exceptions:

- If d1 and d2 both represent Double.NaN, then the equals method returns true, even though Double.NaN==Double.NaN has the value false.
- If d1 represents +0.0 while d2 represents -0.0, or vice versa, the equal test has the value false, even though +0.0==-0.0 has the value true.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.floatValue

public float floatValue()

Returns:

the double value represented by this object is converted to type float and the result of the conversion is returned.

Overrides:

floatValue in class Number (I-§1.11.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.intValue

public int intValue()

Returns:

the double value represented by this object is converted to type int and the result of the conversion is returned.

Overrides:

intValue in class Number (I-§1.11.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.isInfinite

public boolean isInfinite()

Returns:

true if the value represented by this object is positive infinity (I-§1.6.5) or negative infinity (I-§1.6.4); false otherwise.

public static boolean isInfinite(double v)

Parameters:

v- the value to be tested

Returns:

true if the value of the argument is positive infinity (I-§1.6.5) or negative infinity (I-§1.6.4); false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.isNaN

public boolean isNaN()

Returns:

true if the value represented by this object is NaN (I-§1.6.3); false otherwise.

public static boolean isNaN(double v)

Parameters:

v – the value to be tested

Returns:

true if the value of the argument is NaN (I-§1.6.3); false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.longBitsToDouble

public static double longBitsToDouble(long bits)

The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "double precision" bit layout. That floating-point value is returned as the result.

If the argument is 0x7f80000000000000L, the result is positive infinity.

If the argument is 0xff80000000000000L, the result is negative infinity.

If the argument is any value in the range 0x7ff0000000000001L through 0x7fffffffffffffffL or in the range 0xfff0000000000001L through 0xfffffffffffffffL, the result is NaN. All IEEE 754 NaN values are, in effect, lumped together by the Java language into a single value.

Parameters:

`bits` - any long integer

Returns:

the double floating-point value with the same bit pattern.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.longValue

public long longValue()

Returns:

the double value represented by this object is converted to type long and the result of the conversion is returned.

Overrides:

longValue in class Number (I-§1.11.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.toString

public String toString()

The primitive double value represented by this object is converted to a string exactly as if by the method toString of one argument (I-§1.6.21).

Returns:

a String representation of this object.

Overrides:

toString in class Object (I-§1.12.9).

public static String toString(double d)

Creates a string representation of the double argument.

The values NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY, -0.0, and +0.0 are represented by the strings "NaN", "-Infinity", "Infinity", "-0.0" and "0.0", respectively.

If d is in the range $10^{-3} \leq |d| \leq 10^7$, then it is converted to a string in the style [-]ddd.ddd. Otherwise, it is converted to a string in the style [-]m.ddddE±xx.

There is always a minimum of one digit after the decimal point. The number of digits is the minimum needed to uniquely distinguish the argument value from adjacent values of type double.

Parameters:

d – the double to be converted

Returns:

a string representation of the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Double.valueOf

public static Double valueOf(String s)
throws NumberFormatException

Parses a string into a Double.

Parameters:

s – the string to be parsed

Returns:

a newly constructed Double initialized to the value represented by the string argument.

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.7 Class Float

```
public final class java.lang.Float
    extends java.lang.Number (l-§1.11)
{
    // Fields
    public final static float MAX_VALUE; §1.7.1
    public final static float MIN_VALUE; §1.7.2
    public final static float NaN; §1.7.3
    public final static float NEGATIVE_INFINITY; §1.7.4
    public final static float POSITIVE_INFINITY; §1.7.5

    // Constructors
    public Float(double value); §1.7.6
    public Float(float value); §1.7.7
    public Float(String s); §1.7.8

    // Methods
    public double doubleValue(); §1.7.9
    public boolean equals(Object obj); §1.7.10
    public static int floatToIntBits(float value); §1.7.11
    public float floatValue(); §1.7.12
    public int hashCode(); §1.7.13
    public static float intBitsToFloat(int bits); §1.7.14
    public int intValue(); §1.7.15
    public boolean isInfinite(); §1.7.16
    public static boolean isInfinite(float v); §1.7.17
    public boolean isNaN(); §1.7.18
    public static boolean isNaN(float v); §1.7.19
    public long longValue(); §1.7.20
    public String toString(); §1.7.21
    public static String toString(float f); §1.7.22
    public static Float valueOf(String s); §1.7.23
}
```

This class wraps a value of primitive type float in an object. An object of type Float contains a single field whose type is float.

In addition, this class provides a number of methods for converting a float to a String and a String to a float, as well as other constants and methods useful when dealing with a float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.MAX_VALUE

```
public final static float MAX_VALUE = 3.40282346638528860e+38.
```

The largest positive finite value of type float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.MIN_VALUE

```
public final static float MIN_VALUE = 1.40129846432481707e-45
```

The smallest positive value of type float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.NaN

```
public final static float NaN = 0.0f/0.0f
```

The Not-a-Number value of type float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.NEGATIVE_INFINITY

```
public final static float NEGATIVE_INFINITY = -1.0f/0.0f
```

The negative infinity of type float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.POSITIVE_INFINITY

```
public final static float POSITIVE_INFINITY = 1.0f/0.0f
```

The positive infinity of type float.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Float.Float

public Float(double value)

Constructs a newly allocated Float object that represents the argument converted to type float.

Parameters:

value- the value to be represented by the Float

public Float(float value)

Constructs a newly allocated Float object that represents the primitive float argument.

Parameters:

value- the value to be represented by the Float

public Float(String s)
throws NumberFormatException

Constructs a newly allocated Float object that represents the floating-point value of type float represented by the string. The string is converted to a float value as if by the valueOf method (I-§1.7.23).

Parameters:

s- a string to be converted to a Float

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.doubleValue

public double doubleValue()

The float value represented by this object is converted to type double and the result of the conversion is returned.

Overrides:

doubleValue in class Number (I-§1.11.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.equals

public **boolean** **equals**(**Object** obj)

The result is true if the argument is not null and is a Float object that represents a float that has the identical bit pattern to the bit pattern of the float represented by this object.

Note that in most cases, for two instances of class Float, f1 and f2, the value of f1.equals(f2) is true if f1.floatValue() == f2.floatValue() also has the value true. However, there are two exceptions:

- If f1 and f2 both represent Float.NaN, then the equals method returns true, even though Float.NaN==Float.NaN has the value false.
- If f1 represents +0.0f while f2 represents -0.0f, or vice versa, the equal test has the value false, even though 0.0f== -0.0f has the value true.

Returns:

true if the objects are the same; false otherwise.

See Also:

floatToIntBits ([I-§1.7.11](#)).

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.floatToIntBits

`public static int floatToIntBits(float value)`

The result is a representation of the floating-point argument according to the IEEE 754 floating-point "single precision" bit layout.

Bit 31 represents the sign of the floating-point number; bits 30-23 represent the exponent; bits 22-0 represent the significand (sometimes called the mantissa) of the floating-point number.

If the argument is positive infinity, the result is 0x7f800000.

If the argument is negative infinity, the result is 0xff800000.

If the argument is NaN, the result is 0x7fc00000.

Parameters:

`value` - a floating point number

Returns:

the bits that represent the floating point number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.floatValue

public float floatValue()

Returns:

The float value represented by this object is returned.

Overrides:

floatValue in class Number (I-§1.11.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.intBitsToFloat

`public static float intBitsToFloat(int bits)`

The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "single precision" bit layout. That floating-point value is returned as the result.

If the argument is 0x7f800000, the result is positive infinity.

If the argument is 0xff800000, the result is negative infinity.

If the argument is any value in the range 0x7f800001 through 0x7f8fffff or in the range 0xff800001 through 0xff8fffff, the result is NaN. All IEEE 754 NaN values are, in effect, lumped together by the Java language into a single value.

Parameters:

`bits`- an integer

Returns:

the single format floating-point value with the same bit pattern.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.intValue

public int intValue()

Returns:

The float value represented by this object is converted to type int and the result of the conversion is returned.

Overrides:

intValue in class Number (I-§1.11.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.isInfinite

public boolean isInfinite()

Returns:

true if the value represented by this object is positive infinity (I-§1.7.5) or negative infinity (I-§1.7.4); false otherwise.

public static boolean isInfinite(float v)

Parameters:

v – the value to be tested

Returns:

true if the argument is positive infinity (I-§1.7.5) or negative infinity (I-§1.7.4); false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.isNaN

public boolean isNaN()

Returns:

true if the value value represented by this object is NaN (I-§1.7.3); false otherwise.

public static boolean isNaN(float v)

Returns:

true if the argument is NaN (I-§1.7.3); false otherwise.

Parameters:

v– the value to be tested

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.longValue

public long longValue()

Returns:

The float value represented by this object is converted to type long and the result of the conversion is returned.

Overrides:

longValue in class Number (I-§1.11.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.toString

public String toString()

The primitive float value represented by this object is converted to a String exactly as if by the method toString of one argument (I-§1.7.22).

Returns:

a String representation of this object.

Overrides:

toString in class Object (I-§1.12.9).

public static String toString(float f)

Creates a string representation of the float argument.

The values NaN, NEGATIVE_INFINITY, POSITIVE_INFINITY, -0.0, and +0.0 are represented by the strings "NaN", "-Infinity", "Infinity", "-0.0" and "0.0", respectively.

If d is in the range $10^{-3} \leq |d| \leq 10^7$, then it is converted to a String in the style [-]ddd.ddd. Otherwise, it is converted to a string in the style [-]m.ddddE&±xx.

There is always a minimum of one digit after the decimal point. The number of digits is the minimum needed to uniquely distinguish the argument value from adjacent values of type float.

Parameters:

d — the float to be converted

Returns:

a string representation of the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Float.valueOf

`public static Float valueOf(String s)`
`throws NumberFormatException`

Parses a string into a Float.

Parameters:

`s` – the string to be parsed

Returns:

a newly constructed Float initialized to the value represented by the String argument.

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable number.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

§1.8 Class Integer

```
public final class java.lang.Integer
    extends java.lang.Number (l-§1.11)
{
    // Fields
    public final static int MAX_VALUE;      §1.8.1
    public final static int MIN_VALUE;      §1.8.2

    // Constructors
    public Integer(int value);               §1.8.3
    public Integer(String s);               §1.8.4

    // Methods
    public double doubleValue();             §1.8.5
    public boolean equals(Object obj);       §1.8.6
    public float floatValue();              §1.8.7
    public static Integer getInteger(String nm); §1.8.8
    public static Integer getInteger(String nm, int val); §1.8.9
    public static Integer getInteger(String nm, Integer val); §1.8.10
    public int hashCode();                  §1.8.11
    public int intValue();                  §1.8.12
    public long longValue();                §1.8.13
    public static int parseInt(String s);    §1.8.14
    public static int parseInt(String s, int radix); §1.8.15
    public static String toBinaryString(int i); §1.8.16
    public static String toHexString(int i); §1.8.17
    public static String toOctalString(int i); §1.8.18
    public String toString();               §1.8.19
    public static String toString(int i);    §1.8.20
    public static String toString(int i, int radix); §1.8.21
    public static Integer valueOf(String s); §1.8.22
    public static Integer valueOf(String s, int radix); §1.8.23
}
```

This class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides a number of methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Integer.MAX_VALUE

```
public final static int MAX_VALUE = 2147483647
```

The largest value of type int.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Integer.MIN_VALUE

```
public final static int MIN_VALUE = -2147483648
```

The smallest value of type int.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Integer.Integer

public Integer(int value)

Constructs a newly allocated Integer object that represents the primitive int argument.

Parameters:

value- the value to be represented by the Integer

public Integer(String s)
throws NumberFormatException

Constructs a newly allocated Integer object that represents the value represented by the string. The string is converted to an int value as if by the valueOf method (I-§1.8.22).

Parameters:

s- the String to be converted to an Integer

Throws

NumberFormatException (I-§1.41)

If the String does not contain a parsable integer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.doubleValue

public double doubleValue()

Returns:

The int value represented by this object is converted to type double and the result of the conversion is returned.

Overrides:

doubleValue in class Number (I-§1.11.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.equals

`public boolean equals (Object obj)`

The result is true if the argument is not null and is an Integer object that contains the same int value as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.floatValue

public float floatValue()

Returns:

the int value represented by this object is converted to type float and the result of the conversion is returned..

Overrides:

floatValue in class Number (I-§1.11.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.getInteger

public static Integer getInteger(String nm)

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as an integer value and an Integer object representing this value is returned. The full details of the possible numeric formats are given in I-§1.8.10.

If there is no property with the specified name, or if the property does not have the correct numeric format, then null is returned.

Parameters:

nm– property name

Returns:

the Integer value of the property.

public static Integer getInteger(String nm, int val)

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as an integer value and an Integer object representing this value is returned. The full details of the possible numeric formats are given in I-§1.8.10.

If there is no property with the specified name, or if the property does not have the correct numeric format, then an Integer object that represents the value of the second argument is returned.

Parameters:

nm– property name

`val`– default value

Returns:

the Integer value of the property.

`public static Integer getInteger(String nm, Integer val)`

Determines the integer value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` ([I-§1.18.10](#)). The string value of this property is then interpreted as an integer value and an Integer object representing this value is returned.

If the property value begins with "0x" or "#", not followed by a minus sign, the rest of it is parsed as a hexadecimal integer exactly as for the method `Integer.valueOf` ([I-§1.8.23](#)) with radix 16.

If the property value begins with "0" then it is parsed as an octal integer exactly as for the method `Integer.valueOf` ([I-§1.8.23](#)) with radix 8.

Otherwise the property value is parsed as a decimal integer exactly as for the method `Integer.valueOf` ([I-§1.8.23](#)) with radix 10.

The second argument is the default value. If there is no property of the specified name, or if the property does not have the correct numeric format, then the second argument is returned.

Parameters:

`nm`– property name

`val`– default value

Returns:

the Integer value of the property.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Integer.intValue

public int intValue()

Returns:

The int value represented by this object is returned.

Overrides:

intValue in class Number (I-§1.11.3).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Integer.longValue

public long longValue()

Returns:

The int value represented by this object is converted to type long and the result of the conversion is returned.

Overrides:

longValue in class Number (I-§1.11.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.parseInt

public static int parseInt(String s)
throws NumberFormatException

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' to indicate a negative value.

Parameters:

`s` - a string

Returns:

the integer represented by the argument in decimal.

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable integer.

public static int parseInt(String s, int radix)
throws NumberFormatException

Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether Character.digit (I-§1.2.7) returns a nonnegative value), except that the first character may be an ASCII minus sign '-' to indicate a negative value. The resulting integer value is returned.

Parameters:

`s` - the String containing the integer

`radix` - the radix to be used

Returns:

the integer represented by the string argument in the specified radix.

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable integer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.toBinaryString1

public static String toBinaryString(int i)

Creates a string representation of the integer argument as an unsigned integer in base 2.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading zeros.

Parameters:

i – an integer

Returns:

the string representation of the unsigned integer value represented by the argument in binary (base 2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.toHexString₂

`public static String toHexString(int i)`

Creates a string representation of the integer argument as an unsigned integer in base 16.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading zeros.

Parameters:

`i` – an integer

Returns:

the string representation of the unsigned integer value represented by the argument in hexadecimal (base 16).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.toOctalString3

public static String toOctalString(int i)

Creates a string representation of the integer argument as an unsigned integer in base 8.

The unsigned integer value is the argument plus 2^{32} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading zeros.

Parameters:

i – an integer

Returns:

the string representation of the unsigned integer value represented by the argument in octal (base 8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Integer.toString

`public String toString()`

Returns:

a string representation of the value of this object in base 10.

Overrides:

toString in class Object ([I-§1.12.9](#)).

`public static String toString(int i)`

Parameters:

`i` - an integer to be converted

Returns:

a string representation of the argument in base 10.

`public static String toString(int i, int radix)`

Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than `Character.MIN_RADIX` ([I-§1.2.3](#)) or larger than `Character.MAX_RADIX` ([I-§1.2.1](#)), then the radix 10 is used instead.

If the first argument is negative, the first element of the result is the ASCII minus character '-'. If the first argument is not negative, no sign character appears in the result. The following ASCII characters are used as digits:

0123456789abcdefghijklmnopqrstuvwxyz

Parameters:

`i` - an integer

`radix` - the radix

Returns:

a string representation of the argument in the specified radix.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Integer.valueOf

public static Integer valueOf(String s)
throws NumberFormatException

Parameters:

s- the string to be parsed

Returns:

a newly constructed Integer initialized to the value represented by the string argument.

Throws

NumberFormatException (I-§1.41)

If the string does not contain a parsable integer.

public static Integer valueOf(String s, int radix)
throws NumberFormatException

Parameters:

s- the string to be parsed

Returns:

a newly constructed Integer initialized to the value represented by the string argument in the specified radix.

Throws

NumberFormatException (I-§1.41)

If the String does not contain a parsable integer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹This method is new in Java 1.1

²This method is new in Java 1.1.

³This method is new in Java 1.1.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.9 Class Long

```
public final class java.lang.Long
    extends java.lang.Number (l-§1.11)
{
    // Fields
    public final static long MAX_VALUE; §1.9.1
    public final static long MIN_VALUE; §1.9.2

    // Constructors
    public Long(long value); §1.9.3
    public Long(String s); §1.9.4

    // Methods
    public double doubleValue(); §1.9.5
    public boolean equals(Object obj); §1.9.6
    public float floatValue(); §1.9.7
    public static Long getLong(String nm); §1.9.8
    public static Long getLong(String nm, long val); §1.9.9
    public static Long getLong(String nm, Long val); §1.9.10
    public int hashCode(); §1.9.11
    public int intValue(); §1.9.12
    public long longValue(); §1.9.13
    public static long parseLong(String s); §1.9.14
    public static long parseLong(String s, int radix); §1.9.15
    public static String toBinaryString(long i); §1.9.16
    public static String toHexString(long i); §1.9.17
    public static String toOctalString(long i); §1.9.18
    public String toString(); §1.9.19
    public static String toString(long i); §1.9.20
    public static String toString(long i, int radix); §1.9.21
    public static Long valueOf(String s); §1.9.22
    public static Long valueOf(String s, int radix); §1.9.23
}
```

This class wraps a value of the primitive type long in an object. An object of type Long contains a single field whose type is long.

In addition, this class provides a number of methods for converting a long to a String and a String to a long, as well as other constants and methods useful when dealing with a long.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.MAX_VALUE

```
public final static long MAX_VALUE = 9223372036854775807L
```

The largest value of type long.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Long.MIN_VALUE

```
public final static long MIN_VALUE = -9223372036854775808L
```

The smallest value of type long.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Long.Long

public Long(long value)

Constructs a newly allocated Long object that represents the primitive long argument.

Parameters:

value- the value to be represented by the Long

public Long(String s)
throws NumberFormatException

Constructs a newly allocated Long object that represents the value represented by the string. The string is converted to a long value as if by the valueOf method (I-§1.9.22).

Parameters:

s- the string to be converted to a Long

Throws

NumberFormatException (I-§1.41)

If the String does not contain a parsable long integer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.doubleValue

public double doubleValue()

Returns:

The long value represented by this object is converted to type double and the result of the conversion is returned.

Overrides:

doubleValue in class Number (I-§1.11.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.equals

public **boolean** **equals**(**Object** obj)

The result is true if the argument is not null and is a Long object that contains the same long value as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.floatValue

public float floatValue()

Returns:

The long value represented by this object is converted to type float and the result of the conversion is returned.

Overrides:

floatValue in class Number (I-§1.11.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.getLong

public static Long getLong(String nm)

Determines the long value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as a long value and a Long object representing this value is returned. The full details of the possible numeric formats are given in I-§1.9.10.

If there is no property with the specified name, or if the property does not have the correct numeric format, then null is returned.

Parameters:

nm- property name

Returns:

the Long value of the property.

public static Long getLong(String nm, long val)

Determines the long value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as a long value and a Long object representing this value is returned. The full details of the possible numeric formats is given in I-§1.9.10.

If there is no property with the specified name, or if the property does not have the correct numeric format, then a Long object that represents the value of the second argument is returned.

Parameters:

nm- property name

`val`– default value

Returns:

the Long value of the property.

public static Long getLong(String nm, Long val)

Determines the long value of the system property with the specified name.

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (I-§1.18.10). The string value of this property is then interpreted as a long value and a Long object representing this value is returned.

If the property value begins with "0x" or "#", not followed by a minus sign, the rest of it is parsed as a hexadecimal integer exactly as for the method `Long.valueOf` (I-§1.9.23) with radix 16.

If the property value begins with "0" then it is parsed as an octal integer exactly as for the method `Long.valueOf` (I-§1.9.23) with radix 8.

Otherwise the property value is parsed as a decimal integer exactly as for the method `Long.valueOf` (I-§1.9.23) with radix 10.

Note that, in every case, neither L nor l is permitted to appear at the end of the string.

The second argument is the default value. If there is no property of the specified name, or if the property does not have the correct numeric format, then the second argument is returned.

Parameters:

`nm`– the property name

`val`– the default Long value

Returns:

the Long value of the property.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Long.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.intValue

public int intValue()

Returns:

the Long value represented by this object is converted to type int and the result of the conversion is returned.

Overrides:

intValue in class Number (I-§1.11.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.longValue

public long longValue()

Returns:

the Long value represented by this object is returned.

Overrides:

longValue in class Number (I-§1.11.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.parseLong

public static long parseLong(String s)
throws NumberFormatException

Parses the string argument as a signed decimal long. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' to indicate a negative value.

Parameters:

s- a string

Returns:

the long represented by the argument in decimal.

Throws

NumberFormatException ([I-§1.41](#))

If the string does not contain a parsable long.

public static long parseLong(String s, int radix)
throws NumberFormatException

Parses the string argument as a signed long in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether Character.digit ([I-§1.2.7](#)) returns a nonnegative value), except that the first character may be an ASCII minus sign '-' to indicate a negative value. The resulting long value is returned.

Parameters:

s- the String containing the long

radix- the radix to be used

Returns:

the long represented by the string argument in the specified radix.

Throws

NumberFormatException ([I-§1.41](#))

If the string does not contain a parsable integer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.ToString1

public static String ToString(long i)

Creates a string representation of the long argument as an unsigned integer in base 2.

The unsigned long value is the argument plus 2^{64} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading zeros.

Parameters:

i- a long

Returns:

the string representation of the unsigned long value represented by the argument in binary (base 2).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Long.toHexString₂

`public static String toHexString(long i)`

Creates a string representation of the long argument as an unsigned integer in base 16.

The unsigned long value is the argument plus 2^{64} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading zeros.

Parameters:

`i` - a long

Returns:

the string representation of the unsigned long value represented by the argument in hexadecimal (base 16).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Long.ToString3

public static String ToString3(long i)

Creates a string representation of the long argument as an unsigned integer in base 8.

The unsigned long value is the argument plus 2^{64} if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading zeros.

Parameters:

i- a long

Returns:

the string representation of the unsigned long value represented by the argument in octal (base 8).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Long.toString

public String toString()

Returns:

a string representation of this object in base 10.

Overrides:

toString in class Object (I-§1.12.9).

public static String toString(long i)

Parameters:

i- a long to be converted

Returns:

a string representation of the argument in base 10.

public static String toString(long i, int radix)

Creates a string representation of the first argument in the radix specified by the second argument.

If the radix is smaller than Character.MIN_RADIX (I-§1.2.3) or larger than Character.MAX_RADIX (I-§1.2.1), then the radix 10 is used instead.

If the first argument is negative, the first element of the result is the ASCII minus sign '-' . If the first argument is not negative, no sign character appears in the result. The following ASCII characters are used as digits:

0123456789abcdefghijklmnopqrstuvwxyz

Parameters:

i- a long

radix- the radix

Returns:

a string representation of the argument in the specified radix.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Long.valueOf

public static Long valueOf(String s)
throws NumberFormatException

Parameters:

s- the string to be parsed

Returns:

a newly constructed Long initialized to the value represented by the string argument.

Throws

NumberFormatException (I-§1.41)

If the String does not contain a parsable long.

public static Long valueOf(String s, int radix)
throws NumberFormatException

Parameters:

s- the String containing the long

radix- the radix to be used

Returns:

a newly constructed Long initialized to the value represented by the string argument in the specified radix.

Throws

NumberFormatException (I-§1.41)

If the String does not contain a parsable long.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹This method is new in Java 1.1

²This method is new in Java 1.1

³This method is new in Java 1.1

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.10 Class Math

```
public final class java.lang.Math
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static double E; §1.10.1
    public final static double PI; §1.10.2

    // Methods
    public static double abs(double a); §1.10.3
    public static float abs(float a); §1.10.4
    public static int abs(int a); §1.10.5
    public static long abs(long a); §1.10.6
    public static double acos(double a); §1.10.7
    public static double asin(double a); §1.10.8
    public static double atan(double a); §1.10.9
    public static double atan2(double a, double b); §1.10.10
    public static double ceil(double a); §1.10.11
    public static double cos(double a); §1.10.12
    public static double exp(double a); §1.10.13
    public static double floor(double a); §1.10.14
    public static double §1.10.15
        IEEEremainder(double f1, double f2);
    public static double log(double a); §1.10.16
    public static double max(double a, double b); §1.10.17
    public static float max(float a, float b); §1.10.18
    public static int max(int a, int b); §1.10.19
    public static long max(long a, long b); §1.10.20
    public static double min(double a, double b); §1.10.21
    public static float min(float a, float b); §1.10.22
    public static int min(int a, int b); §1.10.23
    public static long min(long a, long b); §1.10.24
    public static double pow(double a, double b); §1.10.25
    public static double random(); §1.10.26
    public static double rint(double a); §1.10.27
    public static long round(double a); §1.10.28
    public static int round(float a); §1.10.29
    public static double sin(double a); §1.10.30
    public static double sqrt(double a); §1.10.31
    public static double tan(double a); §1.10.32
}
```

The class Math contains methods for performing basic numerical operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

To help insure portability of Java programs, the definitions of many of the numerical functions in this

package require that they produce the same results as certain published algorithms. These algorithms are available from the well-known network library netlib as the package fdlibm ("Freely Distributable Math Library"). These algorithms, which are written in the C programming language, are then to be understood as executed with all floating-point operations following the rules of Java floating-point arithmetic.

The network library may be found on the World Wide Web at <http://netlib.att.com> then perform a keyword search for fdlibm.

The Java math library is defined with respect to the version of fdlibm dated 95/01/04. Where fdlibm provides more than one definition for a function (such as acos), the "IEEE754 core function" version is to be used (residing in a file whose name begins with the letter e).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Math.E

```
public final static double E = 2.7182818284590452354
```

The double value that is closer than any other to e , the base of the natural logarithms.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Math.PI

```
public final static double PI = 3.14159265358979323846
```

The double value that is closer than any other to {ewc msdncd, EWGraphic, LAN2j 0 /a "sunref.BMP"}, the ratio of the circumference of a circle to its diameter.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Math.abs

`public static double abs(double a)`

Calculates the absolute value of the argument.

Parameters:

a- a double value

Returns:

the absolute value of the argument1.

`public static float abs(float a)`

Calculates the absolute value of the argument.

Parameters:

a- a float value

Returns:

the absolute value of the argument2.

`public static int abs(int a)`

Calculates the absolute value of the argument. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of Integer.MIN_VALUE (I-§1.8.2), the most negative representable int value, the result is that same value, which is negative.

Parameters:

a- an int value

Returns:

the absolute value of the argument.

public static long **abs**(long a)

Calculates the absolute value of the argument. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of Long.MIN_VALUE (I-§1.9.2), the most negative representable long value, the result is that same value, which is negative.

Parameters:

a- a long value

Returns:

the absolute value of the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.acos

public static double acos(double a)

Parameters:

a- a double value

Returns:

the arc cosine of the argument.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Math.asin

public static double asin(double a)

Parameters:

a- a double value

Returns:

the arc sine of the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.atan

public static double atan(double a)

Parameters:

a- a double value

Returns:

the arc tangent of the argument.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Math.atan2

public static double atan2(double a, double b)

Parameters:

a- a double value

b- a double value

Returns:

the {ewc msdncd, EWGraphic, LAN7j 0 /a "sunref.BMP"} component of the polar coordinate {ewc msdncd, EWGraphic, LAN7j 1 /a "sunref.BMP"} that corresponds to the cartesian coordinate (a, b).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.ceil

public static double ceil(double a)

Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.

Parameters:

a- a double value

Returns:

the value {ewc msdncd, EWGraphic, LAN8j 0 /a "sunref.BMP"}.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.cos

public static double cos(double a)

Parameters:

a— an angle, in radians.

Returns:

the cosine of the argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.exp

public static double exp(double a)

Parameters:

a— a double value

Returns:

the value e^a , where e (I-§1.10.1) is the base of the natural logarithms.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.floor

`public static double floor(double a)`

Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.

Parameters:

a- a double value

Parameters:

a- an assigned value

Returns:

the value {ewc msdncd, EWGraphic, LAN11j 0 /a "sunref.BMP"}.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.IEEEremainder

public static double IEEEremainder(double f1, double f2)

Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard: the remainder value is mathematically equal to $f1 - f2 \text{ times } n$ where n is the mathematical integer closest to the exact mathematical value of the quotient $f1/f2$, and if two mathematical integers are equally close to $f1/f2$ then n is the integer that is even. If the remainder is zero, its sign is the same as the sign of the first argument.

Parameters:

f1- the dividend

f2- the divisor

Returns:

the remainder when f1 is divided by f2.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.log

public static double log(double a)3

Parameters:

a- a number greater than 0.0

Returns:

the value $\ln a$, the natural logarithm of a.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.max

`public static double max(double a, double b)`

Parameters:

a- a double value

b- a double value

Returns:

the larger of a and b⁴.

`public static float max(float a, float b)`

Parameters:

a- a float value

b- a float value

Returns:

the larger of a and b⁵.

`public static int max(int a, int b)`

Parameters:

a- an int value

b- an int value

Returns:

the larger of a and b.

`public static long max(long a, long b)`

Parameters:

a- a long value

b- a long value

Returns:

the larger of a and b.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.min

`public static double min(double a, double b)`

Parameters:

a- a double value

b- a double value

Returns:

the smaller of a and b⁶.

`public static float min(float a, float b)`

Parameters:

a- a float value

b- a float value

Returns:

the smaller of a and b⁷.

`public static int min(int a, int b)`

Parameters:

a- an int value

b- an int value

Returns:

the smaller of a and b.

`public static long min(long a, long b)`

Parameters:

a- a long value

b- a long value

Returns:

the smaller of a and b.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.pow

public static double pow(double a, double b)8

Parameters:

a- a double value

b- a double value

Returns:

the value a^b .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.random

public static double random()

Returns:

a pseudorandom double between 0.0 and 1.0.

See Also:

nextDouble in class Random ([I-§3.7.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math rint

public static double rint(double a)

Calculates the closest integer to the argument.

Parameters:

a- a double value

Returns:

the closest double value to a that is equal to a mathematical integer. If two double values that are mathematical integers are equally close to the value of the argument, the result is the integer value that is even.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.round

public static long round(double a)

Calculates the closest long to the argument.

If the argument is negative infinity or any value less than or equal to the value of Long.MIN_VALUE (I-§1.9.2), the result is equal to the value of Long.MIN_VALUE.

If the argument is positive infinity or any value greater than or equal to the value of Long.MAX_VALUE (I-§1.9.1), the result is equal to the value of Long.MAX_VALUE.

Parameters:

a- a double value

Returns:

the value of the argument rounded to the nearest long value.

public static int round(float a)

Calculates the closest int to the argument.

If the argument is negative infinity or any value less than or equal to the value of Integer.MIN_VALUE (I-§1.8.2), the result is equal to the value of Integer.MIN_VALUE.

If the argument is positive infinity or any value greater than or equal to the value of Integer.MAX_VALUE (I-§1.8.1), the result is equal to the value of Integer.MAX_VALUE.

Parameters:

a- a float value

Returns:

the value of the argument rounded to the nearest int value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.sin

public static double sin(double a)

Parameters:

a- a double value

Returns:

the sine of the argument.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Math.sqrt

public static double sqrt(double a)9

Parameters:

a- a double value

Returns:

the value of {ewc msdncd, EWGraphic, LAN21j 0 /a "sunref.BMP"}. If the argument is NaN or less than zero, the result is NaN.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Math.tan

public static double tan(double a)

Parameters:

a- a double value

Returns:

the tangent of the argument.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, `abs(-0.0)` returns `-0.0`. This bug is fixed in Java 1.1.

²In Java 1.0, `abs(-0.0f)` returns `-0.0f`. This bug is fixed in Java 1.1.

³In Java 1.0, the method `log` was declared as follows

```
public static double log(double a)
throws ArithmeticException
```

even though the `ArithmeticException` was never thrown. This bug is fixed in Java 1.1.

⁴In Java 1.0, `max(-0.0, 0.0)` returns `-0.0`. This bug is fixed in Java 1.1.

⁵In Java 1.0, `max(-0.0f, 0.0f)` returns `-0.0f`. This bug is fixed in Java 1.1.

⁶In Java 1.0, `min(0.0, -0.0)` returns `0.0`. This bug is fixed in Java 1.1.

⁷In Java 1.0, `min(0.0f, -0.0f)` returns `0.0f`. This bug is fixed in Java 1.1.

⁸In Java 1.0, the method `pow` was declared as follows

```
public static double pow(double a, double b)
throws ArithmeticException
```

even though the `ArithmeticException` was never thrown. This bug is fixed in Java 1.1.

⁹In Java 1.0, the method `sqrt` was declared as follows

```
public static double sqrt(double a)
throws ArithmeticException
```

even though the `ArithmeticException` was never thrown. This bug is fixed in Java 1.1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.11 Class Number

```
public abstract class java.lang.Number
    extends java.lang.Object (I-§1.12)
{
    // Methods
    public abstract double doubleValue(); §1.11.1
    public abstract float floatValue(); §1.11.2
    public abstract int intValue(); §1.11.3
    public abstract long longValue(); §1.11.4
}
```

The abstract class Number is the superclass of classes Float (I-§1.7), Double (I-§1.6), Integer (I-§1.8), and Long (I-§1.9).

Subclasses of Number must provide methods to convert the represented numeric value to int, long, float, and double.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Number.doubleValue

public abstract double doubleValue()

Returns:

The numeric value represented by this object is returned after conversion to type double.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Number.floatValue

public abstract float floatValue()

Returns:

The numeric value represented by this object is returned after conversion to type float.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Number.intValue

public abstract int intValue()

Returns:

The numeric value represented by this object is returned after conversion to type int.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Number.longValue

public abstract long longValue()

Returns:

The numeric value represented by this object is returned after conversion to type long.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.12 Class Object

```
public class java.lang.Object
{
    // Constructors
    public Object();    §1.12.1

    // Methods
    protected Object clone(); §1.12.2
    public boolean equals(Object obj); §1.12.3
    protected void finalize();    §1.12.4
    public final Class getClass(); §1.12.5
    public int hashCode();    §1.12.6
    public final void notify();    §1.12.7
    public final void notifyAll(); §1.12.8
    public String toString(); §1.12.9
    public final void wait(); §1.12.10
    public final void wait(long timeout);    §1.12.11
    public final void wait(long timeout, int nanos);    §1.12.12
}
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.Object

public `Object()`

Allocates a new instance of class Object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.clone

protected Object clone()
throws CloneNotSupportedException

The clone method of class Object creates a new object of the same class as this object. It then initializes each of the new object's fields by assigning it the same value as the corresponding field in this object. No constructor is called.

The clone method of class Object will only clone an object whose class indicates that it is willing for its instances to be cloned. A class indicates that its instances can be cloned by declaring that it implements the Cloneable (I-§1.22) interface.

Returns:

a clone of this instance.

Throws

OutOfMemoryError (I-§1.57)

If there is not enough memory.

Throws

CloneNotSupportedException (I-§1.29)

The object's class does not support the Cloneable interface. Subclasses that override the clone method can also throw this exception to indicate that an instance cannot be cloned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.equals

`public boolean equals(Object obj)`

Indicates whether some other object is "equal to" this one.

The equals method implements an equivalence relation:

- It is reflexive: for any reference value x, x.equals(x) should return true.
- It is symmetric: for any reference values x and y, x.equals(y) should return true if y.equals(x) returns true.
- It is transitive: for any reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true then x.equals(z) should return true.
- It is consistent: for any reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false.
- For any reference value x, x.equals(null) should return false.

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any reference values x and y, this method returns true if x and y refer to the same object (x==y has the value true).

Parameters:

obj – the reference object with which to compare

Returns:

true if this object is the same as the obj argument; false otherwise.

See Also:

Hashtable ([I-§3.4](#))

hashCode ([§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.finalize

protected void finalize()
throws Throwable

The finalize method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method in order to dispose of system resources or to perform other cleanup.

Any exception thrown by the finalize method causes the finalization of this object to be halted, but is otherwise ignored.

The finalize method in Object does nothing.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Object.getClass

`public final Class getClass()`

Determines the run-time class of an object.

Returns:

the object of type Class (I-§1.3) that represents the run-time class of the object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.hashCode

public int hashCode()

Calculates a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by java.util.Hashtable (I-§3.4).

The general contract of hashCode is:

- Whenever invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the equals method (I-§1.12.3), then calling the hashCode method on each of the two objects must produce the same integer result.

Returns:

a hash code value for this object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.notify

public final void notify()

Wakes up a single thread that is waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods (I-§1.12.10-§1.12.12).

This method should only be called by a thread that is the owner of this object's monitor. A thread becomes the owner of the object's monitor in one of three ways:

- By executing a synchronized instance method of that object.
- By executing the body of a synchronized statement that synchronizes on the object.
- For objects of type Class, by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.

Throws

IllegalMonitorStateException (I-§1.33)

If the current thread is not the owner of this object's monitor.

See Also:

notifyAll (I-§1.12.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.notifyAll

`public final void notifyAll()`

Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods ([I-§1.12.10-§1.12.12](#)).

This method should only be called by a thread that is the owner of this object's monitor. See the notify method ([I-§1.12.7](#)) for a description of the ways in which a thread can become the owner of a monitor.

Throws

IllegalMonitorStateException ([I-§1.33](#))

If the current thread is not the owner of this object's monitor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.toString

public String toString()

Creates a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read.

The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character '@', and the unsigned hexadecimal representation of the hash code of the object1.

Returns:

A string representation of the object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Object.wait

public final void wait()
throws InterruptedException

Waits to be notified by another thread of a change in this object.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until another thread notifies threads waiting on this object's monitor to wake up either through a call to the notify method ([I-§1.12.7](#)) or the notifyAll method ([I-§1.12.7](#)). The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

This method should only be called by a thread that is the owner of this object's monitor. See the notify method ([I-§1.12.7](#)) for a description of the ways in which a thread can become the owner of a monitor.

Throws

IllegalMonitorStateException ([I-§1.33](#))

If the current thread is not the owner of the object's monitor.

Throws

InterruptedException ([I-§1.37](#))

Another thread has interrupted this thread.

public final void wait(long timeout)
throws InterruptedException

Waits to be notified by another thread of a change in this object.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until either of the following two conditions has occurred:

- Another thread notifies threads waiting on this object's monitor to wake up either through a call to the notify method ([I-§1.12.7](#)) or the notifyAll method ([I-§1.12.7](#)).
- The timeout period, specified by the timeout argument in milliseconds, has elapsed.

The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

This method should only be called by a thread that is the owner of this object's monitor. See the notify method (I-§1.12.7) for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout`– the maximum time to wait in milliseconds

Throws

IllegalMonitorStateException (I-§1.33)

If the current thread is not the owner of the object's monitor.

Throws

InterruptedException (I-§1.37)

Another thread has interrupted this thread.

```
public final void wait(long timeout, int nanos)  
throws InterruptedException
```

Waits to be notified by another thread of a change in this object.

This method is similar to the wait method (I-§1.12.11) of one argument, but it allows finer control over the amount of time to wait for a notification before giving up.

The current thread must own this object's monitor. The thread releases ownership of this monitor and waits until either of the following two conditions has occurred:

- Another thread notifies threads waiting on this object's monitor to wake up either through a call to the notify method (I-§1.12.7) or the notifyAll method (I-§1.12.7).
- The timeout period, specified by timeout milliseconds plus nanos nanoseconds arguments, has elapsed.

The thread then waits until it can re-obtain ownership of the monitor and resumes execution.

This method should only be called by a thread that is the owner of this object's monitor. See the notify method (I-§1.12.7) for a description of the ways in which a thread can become the owner of a monitor.

Parameters:

`timeout`– the maximum time to wait in milliseconds

`nano`– additional time, in nanoseconds range 0-999999

Throws

`IllegalMonitorStateException` ([I-§1.33](#))

If the current thread is not the owner of this object's monitor.

Throws

`InterruptedException` ([I-§1.37](#))

Another thread has interrupted this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the hexadecimal string printed after the '@' is based on the hash code value, but may not be the actual hash code value.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.13 Class Process

```
public abstract class java.lang.Process
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public Process(); §1.13.1

    // Methods
    public abstract void destroy(); §1.13.2
    public abstract int exitValue(); §1.13.3
    public abstract InputStream getErrorStream(); §1.13.4
    public abstract InputStream getInputStream(); §1.13.5
    public abstract OutputStream getOutputStream(); §1.13.6
    public abstract int waitFor(); §1.13.7
}
```

The exec methods (I-§1.14.1-§1.14.4) return an instance of a subclass of Process that can be used to control the process and obtain information about it.

The subprocess is *not* killed when there are no more references to the Process object, but rather the subprocess continues executing asynchronously.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Process.Process

public Process ()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Process.destroy

public abstract void destroy()

Kills the subprocess.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Process.exitValue

public abstract int exitValue()

Returns:

the exit value of the subprocess.

Throws

IllegalThreadStateException (I-§1.34)

If the subprocess has not yet terminated.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Process.getErrorStream

public abstract InputStream **getErrorStream()**

Gets the error stream of the subprocess.

Returns:

the input stream connected to the error stream of the subprocess. This stream is usually unbuffered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Process.getInputStream

public abstract InputStream getInputStream()

Gets the input stream of the subprocess.

Returns:

the input stream connected to the normal output of the subprocess. This stream is usually buffered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Process.getOutputStream

public abstract OutputStream `getOutputStream()`

Gets the output stream of the subprocess.

Returns:

the output stream connected to the normal input of the subprocess. This stream is usually buffered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Process.waitFor

public abstract int waitFor()
throws InterruptedException

Waits for the subprocess to complete. This method returns immediately if the subprocess has already terminated.

Returns:

the exit value of the process.

Throws

InterruptedException (I-§1.37)

If the waitFor was interrupted.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.14 Class Runtime

```
public class java.lang.Runtime
    extends java.lang.Object (I-§1.12)
{
    // Methods
    public Process exec(String command); §1.14.1
    public Process exec(String command, String envp[]); §1.14.2
    public Process exec(String cmdarray[]); §1.14.3
    public Process exec(String cmdarray[], String envp[]); §1.14.4
    public void exit(int status); §1.14.5
    public long freeMemory(); §1.14.6
    public void gc(); §1.14.7
    public InputStream getLocalizedInputStream(InputStream in); §1.14.8
    public OutputStream getLocalizedOutputStream(OutputStream out); §1.14.9
    public static Runtime getRuntime(); §1.14.10
    public void load(String filename); §1.14.11
    public void loadLibrary(String libname); §1.14.12
    public void runFinalization(); §1.14.13
    public long totalMemory(); §1.14.14
    public void traceInstructions(boolean on); §1.14.15
    public void traceMethodCalls(boolean on); §1.14.16
}
```

Every Java application has a single instance of class Runtime which allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the **getRuntime** method (I-§1.14.10).

An application cannot create its own instance of this class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Runtime.exec

public Process **exec**(String command)
throws IOException

Executes the string command in a separate process.

The command argument is parsed into tokens and then executed as a command in a separate process. This method has exactly the same effect as `exec(command, null)` (I-§1.14.2).

Parameters:

command- a specified system command

Returns:

a Process (I-§1.13) object for managing the subprocess.

Throws

SecurityException (I-§1.43)

If the current thread cannot create a subprocess.

public Process **exec**(String command, String envp[])
throws IOException

Executes the string command in a separate process with the specified environment.

This method breaks the command string into tokens and creates a new array cmdarray containing the tokens; it then performs the call `exec(cmdarray, envp)` (I-§1.14.4).

Parameters:

command- a specified system command

envp- array containing environment in format name=value

Returns:

a Process (I-§1.13) object for managing the subprocess.

Throws

SecurityException (I-§1.43)

If the current thread cannot create a subprocess.

```
public Process exec(String cmdarray[])  
throws IOException
```

Executes the command in a separate process with the specified arguments.

The command specified by the tokens in cmdarray is executed as a command in a separate process. This has exactly the same effect as `exec(cmdarray, null)` (I-§1.14.4).

Parameters:

cmdarray- array containing the command to call and its arguments

Returns:

a Process (I-§1.13) object for managing the subprocess.

Throws

SecurityException (I-§1.43)

If the current thread cannot create a subprocess.

```
public Process exec(String cmdarray[], String envp[])  
throws IOException
```

Executes the command in a separate process with the specified arguments and the specified environment.

If there is a security manager, its `checkExec` method (I-§1.15.10) is called with the first component of the array cmdarray as its argument. This may result in a security exception (I-§1.43).

Given an array of strings cmdarray, representing the tokens of a command line, and an array of strings envp, representing an "environment" that defines system properties, this method creates a new process in which to execute the specified command.

Parameters:

cmdarray- array containing the command to call and its arguments

`envp`– array containing environment in format name=value

Returns:

a Process (I-§1.13) object for managing the subprocess.

Throws

SecurityException (I-§1.43)

If the current thread cannot create a subprocess.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.exit

public void exit(int status)

Terminates the currently running Java Virtual Machine. This method never returns normally.

If there is a security manager, its checkExit method ([I-§1.15.11](#)) is called with the status as its argument. This may result in a security exception ([I-§1.43](#)).

The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

Parameters:

status- exit status

Throws

SecurityException ([I-§1.43](#))

If the current thread cannot exit with the specified status.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.freeMemory

public long freeMemory()

Determines the amount of free memory in the system. The value returned by this method is always less than the value returned by the totalMemory method (I-§1.14.14). Calling the gc method (I-§1.14.7) may result in increasing the value returned by freeMemory.

Returns:

an approximation to the total amount of memory currently available for future allocated objects, measured in bytes.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.gc

public void gc()

Calling this method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made its best effort to recycle all unused objects.

The name gc stands for "garbage collector." The Java Virtual Machine performs this recycling process automatically as needed even if the gc method is not invoked explicitly.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.getLocalizedInputStream

public InputStream getLocalizedInputStream(InputStream in)

Creates a localized version of an input stream. This method takes an InputStream (I-§2.13) and returns an InputStream equivalent to the argument in all respects except that it is localized; as characters in the local character set are read from the stream, they are automatically converted from the local character set to Unicode.

If the argument is already a localized stream, it may be returned as the result.

Returns:

a localized input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.getLocalizedOutputStream

public OutputStream getLocalizedOutputStream(OutputStream out)

Creates a localized version of an output stream. This method takes an OutputStream (I-§2.15) and returns an OutputStream equivalent to the argument in all respects except that it is localized; as Unicode characters are written to the stream, they are automatically converted to the local character set.

If the argument is already a localized stream, it may be returned as the result.

Returns:

a localized output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.getRuntime

public static Runtime `getRuntime()`

Returns:

the Runtime object associated with the current Java application.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.load

public void load(String filename)

Loads the given filename as a dynamic library. The filename argument must be a complete path name.

If there is a security manager, its checkLink method (I-§1.15.12) is called with the filename as its argument. This may result in an security exception (I-§1.43).

Parameters:

filename- the file to load

Throws

UnsatisfiedLinkError (I-§1.61)

If the file does not exist.

Throws

SecurityException (I-§1.43)

If the current thread cannot load the specified dynamic library.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.loadLibrary

public void loadLibrary(String libname)

Loads the dynamic library with the specified library name. The mapping from a library name to a specific filename is done in a system-specific manner.

First, if there is a security manager, its checkLink [method \(I-§1.15.12\)](#) is called with the filename as its [argument](#). This may result in an security exception [\(I-§1.43\)](#).

If this [method](#) is called more than once with the same library name, the second and subsequent calls are ignored.

Parameters:

libname- the name of the library

Throws

UnsatisfiedLinkError [\(I-§1.61\)](#)

If the library does not exist.

Throws

SecurityException [\(I-§1.43\)](#)

If the current [thread](#) cannot load the specified dynamic library.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.runFinalization

public void runFinalization()

Calling this method suggests that the Java Virtual Machine expend effort toward running the finalize methods (I-§1.12.4) of objects that have been found to be discarded but whose finalize methods have not yet been run. When control returns from the method call, the Java Virtual Machine has made a best effort to complete all outstanding finalizations.

The Java Virtual Machine performs the finalization process automatically as needed if the runFinalization method is not invoked explicitly.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.totalMemory

public long totalMemory()

Determines the total amount of memory in the Java Virtual Machine.

Returns:

the total amount of memory currently available for allocating objects, measured in bytes.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.traceInstructions

public void traceInstructions(boolean on)

If the boolean argument is true, this method asks the Java Virtual Machine to print out a detailed trace of each instruction in the Java Virtual Machine as it is executed. The virtual machine may ignore this request if it does not support this feature. The destination of the trace output is system-dependent.

If the boolean argument is false, this method causes the Java Virtual Machine to stop performing the detailed instruction trace.

Parameters:

on- true to enable instruction tracing; false to disable this feature

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Runtime.traceMethodCalls

public void traceMethodCalls(boolean on)

If the boolean argument is true, this method asks the Java Virtual Machine to print out a detailed trace of each method in the Java Virtual Machine as it is called. The virtual machine may ignore this request if it does not support this feature. The destination of the trace output is system-dependent.

If the boolean argument is false, this method causes the Java Virtual Machine to stop performing the detailed method trace.

Parameters:

on- true to enable instruction tracing; false to disable this feature

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.15 Class SecurityManager

```
public abstract class java.lang.SecurityManager
    extends java.lang.Object (I-§1.12)
{
    // Fields
    protected boolean inCheck;          §1.15.1

    // Constructors
    protected SecurityManager();          §1.15.2

    // Methods
    public void checkAccept(String host, int port);          §1.15.3
    public void checkAccess(Thread g);          §1.15.4
    public void checkAccess(ThreadGroup g);          §1.15.5
    public void checkConnect(String host, int port);          §1.15.6
    public void checkConnect(String host, int port, §1.15.7
                                Object context);
    public void checkCreateClassLoader();          §1.15.8
    public void checkDelete(String file);          §1.15.9
    public void checkExec(String cmd);          §1.15.10
    public void checkExit(int status);          §1.15.11
    public void checkLink(String lib);          §1.15.12
    public void checkListen(int port);          §1.15.13
    public void checkPackageAccess(String pkg);          §1.15.14
    public void checkPackageDefinition(String pkg);          §1.15.15
    public void checkPropertiesAccess();          §1.15.16
    public void checkPropertyAccess(String key);          §1.15.17
    public void checkRead(FileDescriptor fd);          §1.15.18
    public void checkRead(String file);          §1.15.19
    public void checkRead(String file, Object context);          §1.15.20
    public void checkSetFactory();          §1.15.21
    public boolean checkTopLevelWindow(Object window);          §1.15.22
    public void checkWrite(FileDescriptor fd);          §1.15.23
    public void checkWrite(String file);          §1.15.24
    protected int classDepth(String name);          §1.15.25
    protected int classLoaderDepth();          §1.15.26
    protected ClassLoader currentClassLoader();          §1.15.27
    protected Class[] getClassContext();          §1.15.28
    public boolean getInCheck();          §1.15.29
    public Object getSecurityContext();          §1.15.30
    protected boolean inClass(String name);          §1.15.31
    protected boolean inClassLoader();          §1.15.32
}
```

The security manager is an abstract class that allows applications to implement a security policy. It

allows an application to determine, before performing a possibly unsafe or sensitive operation, what the operation is and whether the operation is being performed by a class created via a class loader ([I-§1.4](#)) rather than installed locally. Classes loaded via a class loader (especially if they have been downloaded over a network) may be less trustworthy than classes from files installed locally. The application has the option of allowing or disallowing the operation.

The `securityManager` class contains a large number of methods whose names begin with the word `check`. These methods are called by various methods in the Java libraries before those methods perform certain potentially sensitive operations. The invocation of such a check method typically looks like this:

```
SecurityManager security = System.getSecurityManager();
if (security != null) {
    security.checkXXX(argument, ... );
}
```

The security manager is thereby given an opportunity to prevent completion of the operation by throwing an exception. A security manager routine simply returns if the operation is permitted, but throws a `SecurityException` ([I-§1.43](#)) if the operation is not permitted. The only exception to this convention is `checkTopLevelWindow` ([I-§1.15.22](#)), which returns a boolean value.

The current security manager is set by the `setSecurityManager` method ([I-§1.18.16](#)) in class `System`. The current security manager is obtained by the `getSecurityManager` method ([I-§1.18.11](#)).

The default implementation of each of the `checkXXX` methods is to assume that the caller does *not* have permission to perform the requested operation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.inCheck

protected boolean inCheck

This field is true if there is a security check in progress; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.SecurityManager

protected `SecurityManager()`

Constructs a new SecurityManager. An application is not allowed to create a new security manager if there is already a current security manager (I-§1.18.11).

Throws

SecurityException (I-§1.43)

If a security manager already exists.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkAccept

public void checkAccept(String host, int port)

This method throws a SecurityException if the calling thread is not permitted to accept a socket connection from the specified host and port number.

This method is invoked for the current security manager (I-§1.18.11) by the accept method (I-§4.5.3) of class ServerSocket.

The checkAccept method for class SecurityManager always throws a SecurityException.

Parameters:

host- the host name of the socket connection

port- the port number of the socket connection

Throws

SecurityException (I-§1.43)

If the caller does not have permission to accept the connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkAccess

public void checkAccess(Thread g)

This method throws a SecurityException if the calling thread is not allowed to modify the thread argument.

This method is invoked for the current security manager (I-§1.18.11) by the stop (I-§1.19.37), suspend (I-§1.19.39), resume (I-§1.19.29), setPriority (I-§1.19.33), setName (I-§1.19.32) and setDaemon (I-§1.19.31) methods of class Thread.

The checkAccess method for class SecurityManager always throws a SecurityException.

Parameters:

g- the thread to be checked

Throws

SecurityException (I-§1.43)

If the caller does not have permission to modify the thread.

public void checkAccess(ThreadGroup g)

This method throws a SecurityException if the calling thread is not allowed to modify the thread group argument.

This method is invoked for the current security manager (I-§1.18.11) when a new child thread or child thread group is created, and by the setDaemon (I-§1.20.18), setMaxPriority (I-§1.20.19), stop (I-§1.20.20), suspend (I-§1.20.21), resume (I-§1.20.17), and destroy (I-§1.20.6) methods of class ThreadGroup.

The checkAccess method for class SecurityManager always throws a SecurityException.

Parameters:

g- the Thread group to be checked

Throws

SecurityException ([I-§1.43](#))

If the caller does not have permission to modify the thread group.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkConnect

public void checkConnect(String host, int port)

This method throws a SecurityException if the calling thread is not allowed to open a socket connection to the specified host and port number.

A port number of -1 indicates that the calling method is attempting to determine the IP address of the specified host name.

The checkConnect method for class SecurityManager always throws a SecurityException.

Parameters:

host- the host name port to connect to

port- the protocol port to connect to

Throws

SecurityException (I-§1.43)

If the caller does not have permission to open a socket connection to the specified host and port.

public void

checkConnect(String host, int port, Object context)

This method throws a SecurityException if the specified security context (I-§1.15.30) is not allowed to open a socket connection to the specified host and port number.

A port number of -1 indicates that the calling method is attempting to determine the IP address of the specified host name.

The checkConnect method for class SecurityManager always throws a SecurityException.

Parameters:

host- the host name port to connect to

`port`- the protocol port to connect to

`context`- a system-dependent security context

Throws

SecurityException ([I-§1.43](#))

If the specified security context does not have permission to open a socket connection to the specified host and port.

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

SecurityManager.checkCreateClassLoader

public void checkCreateClassLoader()

This method throws a SecurityException if the calling thread is not allowed to create a new class loader (I-§1.4.1).

The checkCreateClassLoader method for class SecurityManager always throws a SecurityException.

Throws

SecurityException (I-§1.43)

If the caller does not have permission to create a new class loader.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkDelete

public void checkDelete(String file)

This method throws a SecurityException if the calling thread is not allowed to delete the specified file.

This method is invoked for the current security manager (I-§1.18.11) by the delete method (I-§2.7.10) of class File.

The checkDelete method for class SecurityManager always throws a SecurityException.

Parameters:

file- the system dependent file name

Throws

SecurityException (I-§1.43)

If the caller does not have permission to delete the file.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkExec

public void checkExec(String cmd)

This method throws a SecurityException if the calling thread is not allowed to create a subprocess.

This method is invoked for the current security manager (I-§1.18.11) by the exec methods (§1.14.1-§1.14.4) of class Runtime.

The checkExec method for class SecurityManager always throws a SecurityException.

Parameters:

cmd- the specified system command

Throws

SecurityException (I-§1.43)

If the caller does not have permission to create a subprocess.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkExit

public void checkExit(int status)

This method throws a SecurityException if the calling thread is not allowed to cause the Java Virtual Machine to halt with the specified access code.

This method is invoked for the current security manager (I-§1.18.11) by the exit method (I-§1.14.5) of class Runtime. A status of 0 indicates success; other values indicate various errors.

The checkExit method for class SecurityManager always throws a SecurityException.

Parameters:

status- the exit status

Throws

SecurityException (I-§1.43)

If the caller does not have permission to halt the Java Virtual Machine with the specified status.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkLink

public void checkLink(String lib)

This method throws a SecurityException if the calling thread is not allowed to dynamically link the library code specified by the string argument file. The argument is either a simple library name or a complete file name.

This method is invoked for the current security manager (I-§1.18.11) by methods load (I-§1.14.11) and loadLibrary (I-§1.14.12) of class Runtime.

The checkLink method for class SecurityManager always throws a SecurityException.

Parameters:

lib- the name of the library

Throws

SecurityException (I-§1.43)

If the caller does not have permission to dynamically link the library.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkListen

public void checkListen(int port)

This method throws a SecurityException if the calling thread is not allowed to wait for a connection request on the specified local port number.

The checkListen method for class SecurityManager always throws a SecurityException.

Parameters:

port- the local port

Throws

SecurityException (I-§1.43)

If the caller does not have permission to listen on the specified port.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkPackageAccess

public void checkPackageAccess(String pkg)

This method throws a SecurityException if the calling thread is allowed to access the package specified by the argument.

This method is used by the loadClass method (I-§1.4.4) of class loaders.

The checkPackageAccess method for class SecurityManager always throws a SecurityException.

Parameters:

pkg- the package name

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkPackageDefinition

public void checkPackageDefinition(String pkg)

This method throws a SecurityException if the calling thread is not allowed to define classes in the package specified by the argument.

This method is used by the loadClass method (I-§1.4.4) of some class loaders.

The checkPackageDefinition method for class SecurityManager always throws a SecurityException.

Parameters:

pkg- the package name

Throws

SecurityException (I-§1.43)

If the caller does not have permission to define classes in the specified package.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.checkPropertiesAccess

public void checkPropertiesAccess()

This method throws a SecurityException if the calling thread is not allowed to access or modify the system properties.

This method is used by the getProperties (I-§1.18.8) and setProperties (I-§1.18.15) methods of class System.

The checkPropertiesAccess method for class SecurityManager always throws a SecurityException.

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access or modify the system properties.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkPropertyAccess

public void checkPropertyAccess(String key)

This method throws a SecurityException if the calling thread is not allowed to access the system property with the specified key name.

This method is used by the getProperty (I-§1.18.9) method of class System.

The checkPropertiesAccess method for class SecurityManager always throws a SecurityException.

Parameters:

key- a system property key

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified system property.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkRead

public void checkRead(FileDescriptor fd)

This method throws a SecurityException if the calling thread is not allowed to read from the specified file descriptor (I-§2.8).

The checkRead method for class SecurityManager always throws a SecurityException.

Parameters:

fd- the system-dependent file descriptor

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified file descriptor.

public void checkRead(String file)

This method throws a SecurityException if the calling thread is not allowed to read the file specified by the string argument.

The checkRead method for class SecurityManager always throws a SecurityException.

Parameters:

file- the system dependent file name

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified file.

public void checkRead(String file, Object context)

This method throws a SecurityException if the specified security context is not allowed to read the file specified by the string argument. The context must be a security context returned by a previous call to getSecurityContext (I-§1.15.30).

The checkRead method for class SecurityManager always throws a SecurityException.

Parameters:

`file`- the system dependent file name

`context`- a system-dependent security context

Throws

SecurityException (I-§1.43)

If the specified security context does not have permission to read the specified file.

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

SecurityManager.checkSetFactory

public void checkSetFactory()

This method throws a SecurityException if the calling thread is not allowed to set the socket factor used by ServerSocket ([I-§4.5.7](#)) or Socket ([I-§4.6.11](#)), or the stream handler factory used by URL ([I-§4.8.16](#)).

The checkSetFactory method for class SecurityManager always throws a SecurityException.

Throws

SecurityException ([I-§1.43](#))

The caller does not have permission to specify a socket factory or a stream handler factory.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkTopLevelWindow

public boolean checkTopLevelWindow(Object window)

This method returns false if the calling thread is not trusted to bring up the top-level window indicated by the window argument. In this case, the caller can still decide to show the window, but the window should include some sort of visual warning. If the method returns true, then the window can be shown without any special restrictions.

See class Window ([II-§1.42](#)) for more information on trusted and untrusted windows.

The checkSetFactory method for class SecurityManager always return false.

Parameters:

window- the new window that's being created

Returns:

true if the caller is trusted to put up top-level windows; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.checkWrite

public void checkWrite(FileDescriptor fd)

This method throws a SecurityException if the calling thread is not allowed to write to the specified file descriptor (I-§2.8).

The checkWrite method for class SecurityManager always throws a SecurityException.

Parameters:

fd- the system dependent file descriptor

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified file descriptor.

public void checkWrite(String file)

This method throws a SecurityException if the calling thread is not allowed to write to the file specified by the string argument.

The checkWrite method for class SecurityManager always throws a SecurityException.

Parameters:

file- the system dependent file name

Throws

SecurityException (I-§1.43)

If the caller does not have permission to access the specified file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.classDepth

protected int classDepth(String name)

Determines the stack depth of a given class.

Parameters:

name- the fully qualified name of the class to search for

Returns:

the depth on the stack frame of the first occurrence of a method from a class with the specified name; -1 if such a frame cannot be found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.classLoaderDepth

protected int classLoaderDepth()

Determines the stack of the most recently executing method from a class defined using a class loader.

Returns:

the depth on the stack frame of the most recent occurrence of a method from a class defined using a class loader; returns -1 if there is no occurrence of a method from a class defined using a class loader.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.currentClassLoader

protected `ClassLoader` `currentClassLoader()`

Determines the most recent class loader executing on the stack.

Returns:

the class loader of the most recent occurrence on the stack of a method from a class defined using a class loader; returns null if there is no occurrence on the stack of a method from a class defined using a class loader.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.getClassContext

protected `Class[] getClassContext()`

Calculates the current execution stack, which is returned as an array of classes.

The length of the array is the number of methods on the execution stack. The element at index 0 is the class of the currently executing method. The element at index 1 is the class of that method's caller, and so forth.

Returns:

the execution stack.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.getInCheck

public boolean getInCheck()

Returns:

the value of the inCheck (I-§1.15.1) field. This field should contain true if a security check is in progress; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.getSecurityContext

public Object getSecurityContext()

Creates an object which encapsulates the current execution environment. The result of this method is used by the three-argument checkConnect method (I-§1.15.7) and by the two-argument checkRead method (I-§1.15.20).

These methods are needed because a trusted method may be called upon to read a file or open a socket on behalf of another method. The trusted method needs to determine if the other (possibly untrusted) method would be allowed to perform the operation on its own.

Returns:

an implementation-dependent object which encapsulates sufficient information about the current execution environment to perform some security checks later.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SecurityManager.inClass

protected boolean inClass(String name)

Parameters:

name- the fully-qualified name of the class

Returns:

true if a method from a class with the specified name is on the execution stack; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityManager.inClassLoader

protected boolean inClassLoader()

Returns:

true if a method from a class defined using a class loader is on the execution stack.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.16 Class String

```
public final class java.lang.String
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public String(); §1.16.1
    public String(byte ascii[], int hiByte); §1.16.2
    public String(byte ascii[], int hiByte, §1.16.3
                  int offset, int count);
    public String(char value[]); §1.16.4
    public String(char value[], int offset, int count); §1.16.5
    public String(String value); §1.16.6
    public String(StringBuffer buffer); §1.16.7

    // Methods
    public char charAt(int index); §1.16.8
    public int compareTo(String anotherString); §1.16.9
    public String concat(String str); §1.16.10
    public static String copyValueOf(char data[]); §1.16.11
    public static String §1.16.12
        copyValueOf(char data[], int offset, int count);
    public boolean endsWith(String suffix); §1.16.13
    public boolean equals(Object anObject); §1.16.14
    public boolean equalsIgnoreCase(String anotherString); §1.16.15
    public void getBytes(int srcBegin, int srcEnd, §1.16.16
                        byte dst[], int dstBegin);
    public void getChars(int srcBegin, int srcEnd, §1.16.17
                        char dst[], int dstBegin);

    public int hashCode(); §1.16.18
    public int indexOf(int ch); §1.16.19
    public int indexOf(int ch, int fromIndex); §1.16.20
    public int indexOf(String str); §1.16.21
    public int indexOf(String str, int fromIndex); §1.16.22
    public String intern(); §1.16.23
    public int lastIndexOf(int ch); §1.16.24
    public int lastIndexOf(int ch, int fromIndex); §1.16.25
    public int lastIndexOf(String str); §1.16.26
    public int lastIndexOf(String str, int fromIndex); §1.16.27

    public int length(); §1.16.28
    public boolean regionMatches(boolean ignoreCase, §1.16.29
                                int toffset, String other, int ooffset, int len);
    public boolean regionMatches(int toffset, String other, §1.16.30
                                int ooffset, int len);
    public String replace(char oldChar, §1.16.31
```



```

        char newChar);
    public boolean startsWith(String prefix); §1.16.32
    public boolean startsWith(String prefix, int toffset); §1.16.33
    public String substring(int beginIndex); §1.16.34
    public String substring(int beginIndex, int endIndex); §1.16.35
    public char[] toCharArray(); §1.16.36
    public String toLowerCase(); §1.16.37
    public String toString(); §1.16.38
    public String toUpperCase(); §1.16.39
    public String trim(); §1.16.40
    public static String valueOf(boolean b); §1.16.41
    public static String valueOf(char c); §1.16.42
    public static String valueOf(char data[]); §1.16.43
    public static String §1.16.44
        valueOf(char data[], int offset, int count);
    public static String valueOf(double d); §1.16.45
    public static String valueOf(float f); §1.16.46
    public static String valueOf(int i); §1.16.47
    public static String valueOf(long l); §1.16.48
    public static String valueOf(Object obj); §1.16.49
}

```

The String class represents character strings. All string literals in Java programs, such as "abc" are implemented as instances of this class.

Strings are constant, their values cannot be changed after they are created. String buffers (l-§1.17) support mutable strings.

The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.String

public String()

Allocates a new String containing no characters.

public String(byte ascii[], int hibyte)

Allocates a new String containing characters constructed from an array of 8-bit integer values. Each character *c* in the resulting string is constructed from the corresponding component *b* in the byte array such that:

$$c == (\text{char})(((\text{hibyte} \& 0\text{xff}) \ll 8) \mid (\text{b} \& 0\text{xff}))$$

Parameters:

ascii- the bytes to be converted to characters

hibyte- the top 8 bits of each 16-bit Unicode character

public String(byte ascii[], int hibyte, int offset, int count)

Allocates a new String constructed from a subarray of an array of 8-bit integer values.

The offset argument is the index of the first byte of the subarray, and the count argument specifies the length of the subarray.

Each byte in the subarray is converted to a char as specified in the method above (I-§1.16.2).

Parameters:

ascii- the bytes to be converted to characters

hibyte- the top 8 bits of each 16-bit Unicode character

`offset`- the initial offset

`count`- the length

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset or count argument is invalid.

public String(char value[])

Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

Parameters:

`value`- the initial value of the string

public String(char value[], int offset, int count)

Allocates a new String that contains characters from a subarray of the character array argument. The offset argument is the index of the first character of the subarray and the count argument specifies the length of the subarray.

Parameters:

`value`- array that is the source of characters

`offset`- the initial offset

`count`- the length

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset and count arguments index characters outside the bounds of the value array.

public String(String value)

Allocates a new string that contains the same sequence of characters as the string argument.

Parameters:

value- a String

public String(StringBuffer buffer)

Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Parameters:

buffer- a StringBuffer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.charAt

public char charAt(int index)

Returns:

The character at the specified index of this string. The first character is at index 0.

Parameters:

index- the index of the desired character

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the index is out of range.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.CompareTo

```
public int CompareTo(String anotherString)
```

Compares two strings lexicographically.

Parameters:

`anotherString`– the String to be compared

Returns:

The value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.concat

public String concat(String str)

Concatenates the string argument to the end of this string.

If the length of the argument string is zero, then this object is returned.

Parameters:

str- the String which is concatenated to the end of this String

Returns:

A string that represents the concatenation of this object's characters followed by the string argument's characters.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.copyOfValueOf

```
public static String copyValueOf(char data[])
```

Returns:

a String that contains the characters of the character array.

Parameters:

data- the character array

```
public static String
```

```
copyValueOf(char data[], int offset, int count)
```

Parameters:

data- the character array

offset- initial offset of the subarray

count- length of the subarray

Returns:

a String that contains the characters of the specified subarray of the character array.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.endsWith

public boolean endsWith(String suffix)

Parameters:

suffix- the suffix

Returns:

true if the character sequence represented by the argument is a suffix of the character sequence represented by this object; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.equals

public **boolean** **equals**(**Object** **anObject**)

The result is true if the argument is not null and is a String object that represents the same sequence of characters as this object.

Parameters:

anObject– the object to compare this String against

Returns:

true if the String's are equal; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

See Also:

equalsIgnoreCase ([I-§1.16.15](#))

compareTo ([I-§1.16.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.EqualsIgnoreCase

public **boolean** EqualsIgnoreCase(**String** anotherString)

The result is true if the argument is not null and is a **String** object that represents the same sequence of characters as this object, where case is ignored.

Two characters are considered the same, ignoring case, if at least one of the following is true:

- The two characters are the same (as compared by the == operator).
- Applying the method Character.ToUppercase (I-§1.2.24) to each character produces the same result.
- Applying the method Character.ToLowercase (I-§1.2.21) to each character produces the same result.

Two sequence of characters are the same, ignoring case, if the sequences have the same length and corresponding characters are the same, ignoring case.

Parameters:

anotherString- the String to compare this String against

Returns:

true if the String's are equal, ignoring case; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.getBytes

```
public void getBytes(int srcBegin, int srcEnd, byte dst[], int  
dstBegin)
```

Copies characters from this string into the destination byte array. Each byte receives the 8 low-order bits of the corresponding character.

The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1. The total number of characters to be copied is srcEnd-srcBegin. The characters, converted to bytes, are copied into the subarray of dst starting at index dstBegin and ending at index:

$\text{dstbegin} + (\text{srcEnd} - \text{srcBegin}) - 1$

Parameters:

srcBegin- index of the first character in the string to copy

srcEnd- index after the last character in the string to copy

dst- the destination array

dstBegin- the start offset in the destination array

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.getChars

```
public void getChars(int srcBegin, int srcEnd, char dst[], int  
dstBegin)
```

Copies characters from this string into the destination character array.

The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1 (thus the total number of characters to be copied is srcEnd-srcBegin). The characters are copied into the subarray of dst starting at index dstBegin and ending at index:

$\text{dstbegin} + (\text{srcEnd} - \text{srcBegin}) - 1$

Parameters:

srcBegin- index of the first character in the string to copy

srcEnd- index after the last character in the string to copy

dst- the destination array

dstBegin- the start offset in the destination array

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.indexOf

public int indexOf(int ch)

Parameters:

ch- a character

Returns:

the index of the first occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur.

public int indexOf(int ch, int fromIndex)

Parameters:

ch- a character

fromIndex- the index to start the search from

Returns:

the index of the first occurrence of the character in the character sequence represented by this object that is greater than or equal to fromIndex, or -1 if the character does not occur.

public int indexOf(String str)

Parameters:

str- any string

Returns:

if the string argument occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.

public int indexOf(String str, int fromIndex)

Parameters:

str- the substring to search for

fromIndex- the index to start the search from

Returns:

If the string argument occurs as a substring within this object at a starting index no smaller than fromIndex then the index of the first character of the first such substring is returned. If it

does not occur as a substring starting at fromIndex or beyond, -1 is returned.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.intern

public String intern()

Creates a canonical representation for the string object.

If s and t are strings such that s.equals(t), it is guaranteed that s.intern() == t.intern().

Returns:

a string that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.lastIndexOf

```
public int lastIndexOf(int ch)
```

Parameters:

ch- a character

Returns:

the index of the last occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur.

```
public int lastIndexOf(int ch, int fromIndex)
```

Parameters:

ch- a character

fromIndex- the index to start the search from

Returns:

the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to from-Index, or -1 if the character does not occur before that point.

```
public int lastIndexOf(String str)
```

Parameters:

str- the substring to search for

Returns:

If the string argument occurs as a substring within this object, then the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.

```
public int lastIndexOf(String str, int fromIndex)
```

Parameters:

str- the substring to search for

fromIndex- the index to start the search from

Returns:

If the string argument occurs as a substring within this object at a starting index no greater

than fromIndex then the index of the first character of the last such substring is returned. If it does not occur as a substring starting at fromIndex or earlier, -1 is returned.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.length

public int length()

Returns:

the length of the sequence of characters represented by this object is returned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.regionMatches

public boolean

```
regionMatches(boolean ignoreCase, int toffset, String other, int  
ooffset, int len)
```

Determines if two string regions are equal.

If toffset or ooffset is negative, or if toffset+length is greater than the length of this string, or if ooffset+length is greater than the length of the string argument, then this method returns false.

Parameters:

ignoreCase- if true, ignore case when comparing characters

toffset- starting offset of the subregion in this string

other- string argument

ooffset- starting offset of the subregion in the string argument

len- the number of characters to compare

Returns:

true if the specified subregion of this string matches the specified subregion of the string argument; false otherwise. Whether the matching is exact or case insensitive depends on the ignoreCase argument.

public boolean

```
regionMatches(int toffset, String other,  
               int ooffset, int len)
```

Determines if two string regions are equal.

If toffset or ooffset is negative, or if toffset+length is greater than the length of this string, or if ooffset+length is greater than the length of the string argument, then this method returns false.

Parameters:

`toffset`- starting offset of the subregion in this string

`other`- string argument

`ooffset`- starting offset of the subregion in the string argument

`len`- the number of characters to compare

Returns:

true if the specified subregion of this string exactly matches the specified subregion of the string argument; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.replace

public String replace(char oldChar, char newChar)

Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

If the character oldChar does not occur in the character sequence represented by this object, then this string is returned.

Parameters:

oldChar- the old character

newChar- the new character

Returns:

a string derived from this string by replacing every occurrence of oldChar with newChar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.startsWith

```
public boolean startsWith(String prefix)
```

Parameters:

`prefix`- the prefix

Returns:

true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

```
public boolean startsWith(String prefix, int toffset)
```

Parameters:

`prefix`- the prefix

`toffset`- where to begin looking in the the String

Returns:

true if the character sequence represented by the argument is a prefix of the substring of this object starting at index toffset; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


String.substring

public String substring(int beginIndex)

Creates a new string that is a substring of this string. The substring begins at the specified index and extends to the end of this string.

Parameters:

beginIndex- the beginning index, inclusive

Returns:

the specified substring.

Throws

StringIndexOutOfBoundsException ([I-§1.44](#))

If the beginIndex is out of range.

public String substring(int beginIndex, int endIndex)

Creates a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1.

Parameters:

beginIndex- the beginning index, inclusive

endIndex- the ending index, exclusive

Returns:

the specified substring.

Throws

StringIndexOutOfBoundsException ([I-§1.44](#))

If the beginIndex or the endIndex is out of range.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.toCharArray

public char[] toCharArray()

Returns:

A newly allocated character array whose length is the length of this string and whose contents is initialized to contain the character sequence represented by this string.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.ToLowerCase

public String toLowerCase()

Converts a string to lowercase.

If no character in this string has a different lowercase version (I-§1.2.21), then this string is returned.

Otherwise, a new string is allocated, whose length is identical to this string, and such that each character which has a difference lowercase version is mapped to this lower case equivalent.

Returns:

the string, converted to lowercase.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.toString

public String toString()

This object is itself returned.

Returns:

the string itself.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.toUpperCase

public String toUpperCase()

Converts a string to uppercase.

If no character in this string has a different uppercase version (I-§1.2.24), then this string is returned.

Otherwise, a new string is allocated, whose length is identical to this string, and such that each character which has a difference uppercase version is mapped to this uppercase equivalent.

Returns:

the string, converted to uppercase.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.trim

public String trim()

Removes whitespace from both ends of a string.

All characters that have codes less than or equal to '\u0020' (the space character) are considered to be whitespace.

Returns:

this string, with whitespace removed from the front and end.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

String.valueOf

`public static String valueOf(boolean b)`

Creates the string representation of the boolean argument.

Parameters:

b- a boolean

Returns:

if the argument is true, a string equal to "true" is returned; otherwise, a string equal to "false" is returned.

`public static String valueOf(char c)`

Creates the string representation of the char argument.

Parameters:

c- a char

Returns:

a newly allocated string of length one containing as its single character the argument c.

`public static String valueOf(char data[])`

Creates the string representation of the char array argument.

Parameters:

data- a char array

Returns:

a newly allocated string representing the same sequence of characters contained in the character array argument.

`public static String`

`valueOf(char data[], int offset, int count)`

Creates the string representation of a specific subarray of the char array argument.

The offset argument is the index of the first character of the subarray. The count argument specifies the length of the subarray.

Parameters:

`data`– the character array

`offset`– the initial offset into the value of the String

`count`– the length of the value of the String

Returns:

a newly allocated string representing the sequence of characters contained in the subarray of the character array argument.

public static String valueOf(double d)

Creates the string representation of the double argument.

The representation is exactly the one returned by the Double.toString method of one argument (I-§1.6.21).

Parameters:

`d`– a double

Returns:

a newly allocated string containing a string representation of the double argument.

public static String valueOf(float f)

Creates the string representation of the float argument.

The representation is exactly the one returned by the Float.toString method of one argument (I-§1.7.22).

Parameters:

f– a float

Returns:

a newly allocated string containing a string representation of the float argument.

public static String valueOf(int i)

Creates the string representation of the int argument.

The representation is exactly the one returned by the Integer.toString method of one argument (I-§1.8.20).

Parameters:

i– an int

Returns:

a newly allocated string containing a string representation of the int argument.

public static String valueOf(long l)

Creates the string representation of a the long argument.

The representation is exactly the one returned by the Long.toString method of one argument (I-§1.9.20).

Parameters:

l– a long

Returns:

a newly allocated string containing a string representation of the long argument.

public static String valueOf(Object obj)

Creates the string representation of the Object argument.

Parameters:

`obj` – an Object

Returns:

If the argument is null, then a string equal to "null"; otherwise the value of `obj.toString()` is returned.

See Also:

`toString` in class `Object` ([I-§1.12.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.17 Class StringBuffer

```
public class java.lang.StringBuffer
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public StringBuffer(); §1.17.1
    public StringBuffer(int length); §1.17.2
    public StringBuffer(String str); §1.17.3

    // Methods
    public StringBuffer append(boolean b); §1.17.4
    public StringBuffer append(char c); §1.17.5
    public StringBuffer append(char str[]); §1.17.6
    public StringBuffer §1.17.7
        append(char str[], int offset, int len);
    public StringBuffer append(double d); §1.17.8
    public StringBuffer append(float f); §1.17.9
    public StringBuffer append(int i); §1.17.10
    public StringBuffer append(long l); §1.17.11
    public StringBuffer append(Object obj); §1.17.12
    public StringBuffer append(String str); §1.17.13
    public int capacity(); §1.17.14
    public char charAt(int index); §1.17.15
    public void ensureCapacity(int minimumCapacity); §1.17.16
    public void getChars(int srcBegin, int srcEnd, §1.17.17
        char dst[], int dstBegin);
    public StringBuffer insert(int offset, boolean b); §1.17.18
    public StringBuffer insert(int offset, char c); §1.17.19
    public StringBuffer insert(int offset, char str[]); §1.17.20
    public StringBuffer insert(int offset, double d); §1.17.21
    public StringBuffer insert(int offset, float f); §1.17.22
    public StringBuffer insert(int offset, int i); §1.17.23
    public StringBuffer insert(int offset, long l); §1.17.24
    public StringBuffer insert(int offset, Object obj); §1.17.25
    public StringBuffer insert(int offset, String str); §1.17.26
    public int length(); §1.17.27
    public StringBuffer reverse(); §1.17.28
    public void setCharAt(int index, char ch); §1.17.29
    public void setLength(int newLength); §1.17.30
    public String toString(); §1.17.31
}
```

A string buffer implements a mutable sequence of characters.

String buffers are safe for use by multiple threads. The methods are synchronized where necessary

so that all the operations on any particular instance behave as if they occur in some serial order.

String buffers are used by the compiler to implement the binary string concatenation operator +. For example the code

```
x = "a" + 4 + "c"
```

is compiled to the equivalent of:

```
x = new StringBuffer().append("a").append(4).append("c").toString()
```

The principal operations on a StringBuffer are the append and insert methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string buffer. The append method always adds these characters at the end of the buffer; the insert method adds the characters at a specified point.

For example, if z refers to a string buffer object whose current contents are "start" then the method call z.append("le") would cause the string buffer to contain "startle" while z.insert(4, "le") would alter the string buffer to contain "starlet".

Every string buffer has a capacity. As long as the length of the character sequence contained in the string buffer does not exceed the capacity, it is not necessary to allocate a new internal buffer array. If the internal buffer overflows, it is automatically made larger.

See Also:

String (I-§1.16)

ByteArrayOutputStream (I-§2.4).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBuffer.StringBuffer

public StringBuffer()

Constructs a string buffer with no characters in it, and an initial capacity of 16 characters.

public StringBuffer(int length)

Constructs a string buffer with no characters in it, and an initial capacity specified by the length argument.

Parameters:

length- the initial capacity

Throws

NegativeArraySizeException ([I-§1.38](#))

If the length argument is less than zero.

public StringBuffer(String str)

Constructs a string buffer so that it represents the same sequence of characters as the string argument. The initial capacity of the string buffer is 16 plus the length of the string argument.

Parameters:

str- the initial contents of the buffer

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBuffer.append

public **StringBuffer** **append**(boolean b)

Appends the string representation of the boolean argument to the string buffer.

The argument is converted to a string as if by the method `String.valueOf` ([I-§1.16.41](#)), and the characters of that string are then appended ([I-§1.17.13](#)) to this string buffer.

Parameters:

b- a boolean

Returns:

this string buffer.

public **StringBuffer** **append**(char c)

Appends the string representation of the char argument to this string buffer.

The argument is appended to the contents of this string buffer. The length of this string buffer increases by one.

Parameters:

ch- a char

Returns:

this string buffer.

public **StringBuffer** **append**(char str[])

Appends the string representation of the char array argument to this string buffer.

The characters of the array argument are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the length of the argument.

Parameters:

`str`– the characters to be appended

Returns:

this string buffer.

```
public StringBuffer append(char str[], int offset, int len)
```

Appends the string representation of a subarray of the char array argument to this string buffer.

Characters of the character array `str`, starting at index offset, are appended, in order, to the contents of this string buffer. The length of this string buffer increases by the value of `len`.

Parameters:

`str`– the characters to be appended

`offset`– index of first character to append

`len`– the number of characters to append

Returns:

this string buffer.

```
public StringBuffer append(double d)
```

Appends the string representation of the double argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf` ([I-§1.16.45](#)), and the characters of that string are then appended ([I-§1.17.13](#)) to this string buffer.

Parameters:

`d`– a double

Returns:

this string buffer.

public **StringBuffer** **append(float f)**

Appends the string representation of the float argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf` (I-§1.16.46), and the characters of that string are then appended (I-§1.17.13) to this string buffer.

Parameters:

f– a float

Returns:

this string buffer.

public **StringBuffer** **append(int i)**

Appends the string representation of the int argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf` (I-§1.16.47), and the characters of that string are then appended (I-§1.17.13) to this string buffer.

Parameters:

i– an int

Returns:

this string buffer.

public **StringBuffer** **append(long l)**

Appends the string representation of the long argument to this string buffer.

The argument is converted to a string as if by the method `String.valueOf` (I-§1.16.48), and the characters of that string are then appended (I-§1.17.13) to this string buffer.

Parameters:

l- a long

Returns:

this string buffer.

public StringBuffer append(Object obj)

Appends the string representation of the Object argument to this string buffer.

The argument is converted to a string as if by the method String.valueOf (I-§1.17.3), and the characters of that string are then appended (I-§1.17.13) to this string buffer.

Parameters:

obj- an Object

Returns:

this string buffer.

public StringBuffer append(String str)

Appends the string to this string buffer.

The characters of the String argument are appended, in order, to the contents of this string buffer, increasing the length of this string buffer by the length of the argument.

Parameters:

str- a string

Returns:

this string buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.capacity

public int capacity()

Returns:

the current capacity of this string buffer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBuffer.charAt

public char charAt(int index)

Determines the character at a specific index in this string buffer.

The first character of a string buffer is at index 0, the next at index 1, and so on, for array indexing.

The index argument must be greater than or equal to 0, and less than the length (I-§1.17.27) of this string buffer.

Parameters:

index- the index of the desired character

Returns:

the character at the specified index of this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the index is invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.ensureCapacity

public void ensureCapacity(int minimumCapacity)

If the current capacity of this string buffer is less than the argument, then a new internal buffer is allocated with greater capacity. The new capacity is the larger of:

- the minimumCapacity argument
- twice the old capacity, plus 2

If the minimumCapacity argument is nonpositive, this method takes no action and simply returns.

Parameters:

minimumCapacity- the minimum desired capacity

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.getChars

```
public void getChars(int  srcBegin, int  srcEnd, char  dst[], int  
dstBegin)
```

Characters are copied from this string buffer into the destination character array dst. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1. The total number of characters to be copied is srcEnd-srcBegin. The characters are copied into the subarray of dst starting at index dstBegin and ending at index dstbegin+(srcEnd-srcBegin)-1.

Parameters:

srcBegin- begin copy at this offset in the string buffer

srcEnd- stop copying at this offset in the string buffer

dst- the array to copy the data into

dstBegin- offset into dst

Throws

StringIndexOutOfBoundsException (I-§1.44)

If there is an invalid index into the buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.insert

```
public StringBuffer insert(int offset, boolean b)
```

Inserts the string representation of the boolean argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf` ([I-§1.16.41](#)), and the characters of that string are then inserted ([I-§1.17.26](#)) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length ([I-§1.17.27](#)) of this string buffer.

Parameters:

`offset`- the offset

`b`- a boolean

Returns:

this string buffer.

Throws

`StringIndexOutOfBoundsException` ([I-§1.44](#))

If the offset is invalid.

```
public StringBuffer insert(int offset, char c)
```

Inserts the string representation of the char argument into this string buffer.

The second argument is inserted into the contents of this string buffer at the position indicated by offset. The length of this string buffer increases by one.

The offset argument must be greater than or equal to 0, and less than or equal to the length ([I-§1.17.27](#)) of this string buffer.

Parameters:

`offset`- the offset

`ch`- a char

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException ([I-§1.44](#))

If the offset is invalid.

public StringBuffer insert(int offset, char str[])

Inserts the string representation of the char array argument into this string buffer.

The characters of the array argument are inserted into the contents of this string buffer at the position indicated by offset. The length of this string buffer increases by the length of the argument.

Parameters:

`offset`- the offset

`ch`- a character array

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException ([I-§1.44](#))

If the offset is invalid.

public StringBuffer insert(int offset, double d)

Inserts the string representation of the double argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf` ([I-§1.16.45](#)), and the characters of that string are then inserted ([I-§1.17.26](#)) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

`offset`– the offset

`b`– a double

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset is invalid.

public StringBuffer insert(int offset, float `f`)

Inserts the string representation of the float argument into this string buffer.

The second argument is converted to a string as if by the method String.valueOf (I-§1.16.46), and the characters of that string are then inserted (I-§1.17.26) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

`offset`– the offset

`b`– a float

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset is invalid.


```
public StringBuffer insert(int offset, int i)
```

Inserts the string representation of the second int argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf` (I-§1.16.47), and the characters of that string are then inserted (I-§1.17.26) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

`offset`– the offset

`i`– an int

Returns:

this string buffer.

Throws

`StringIndexOutOfBoundsException` (I-§1.44)

If the offset is invalid.

```
public StringBuffer insert(int offset, long l)
```

Inserts the string representation of the long argument into this string buffer.

The second argument is converted to a string as if by the method `String.valueOf` (I-§1.16.48), and the characters of that string are then inserted (I-§1.17.26) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

`offset`– the offset

b- a long

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset is invalid.

public StringBuffer insert(int offset, Object obj)

Inserts the string representation of the Object argument into this string buffer.

The second argument is converted to a string as if by the method String.valueOf (I-§1.16.49), and the characters of that string are then inserted (I-§1.17.26) into this string buffer at the indicated offset.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

offset- the offset

b- an Object

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset is invalid.

public StringBuffer insert(int offset, String str)

Inserts the string into this string buffer.

The characters of the String argument are inserted, in order, into this string buffer at the indicated offset. The length of this string buffer is increased by the length of the argument.

The offset argument must be greater than or equal to 0, and less than or equal to the length (I-§1.17.27) of this string buffer.

Parameters:

`offset`- the offset

`str`- a string

Returns:

this string buffer.

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the offset is invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.length

public int length()

Returns:

the number of characters in this string buffer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBuffer.reverses1

public `StringBuffer reverse()`

The character sequence contained in this string buffer is replaced by the reverse of the sequence.

Returns:

this string buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.setCharAt

```
public void setCharAt(int index, char ch)
```

The character at the specified index of this string buffer is set to ch.

The offset argument must be greater than or equal to 0, and less than the length (I-§1.17.27) of this string buffer.

Parameters:

`index`- the index of the character to modify

`ch`- the new character

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the index is invalid.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBuffer.setLength

public void setLength(int newLength)

If the newLength argument is less than the current length (I-§1.17.27) of the string buffer, the string buffer is truncated to contain exactly the number of characters given by the newLength argument.

If the newLength argument is greater than or equal to the current length, sufficient null characters ('\u0000') are appended to the string buffer so that length becomes the newLength argument.

The newLength argument must be greater than or equal to zero.

Parameters:

newLength- the new length of the buffer

Throws

StringIndexOutOfBoundsException (I-§1.44)

If the newLength argument is invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBuffer.toString

public String toString()

A new String object is allocated and initialized to contain the character sequence currently represented by this string buffer. This String is then returned. Subsequent changes to the string buffer do not affect the contents of the String.

Returns:

a string representation of the string buffer.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹This method is new in Java 1.1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.18 Class System

```
public final class java.lang.System
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public static PrintStream err; §1.18.1
    public static InputStream in; §1.18.2
    public static PrintStream out; §1.18.3

    // Methods
    public static void §1.18.4
        arraycopy(Object src, int src_position,
            Object dst, int dst_position, int length);
    public static long currentTimeMillis(); §1.18.5
    public static void exit(int status); §1.18.6
    public static void gc(); §1.18.7
    public static Properties getProperties(); §1.18.8
    public static String getProperty(String key); §1.18.9
    public static String getProperty(String key, String def); §1.18.10
    public static SecurityManager getSecurityManager(); §1.18.11
    public static void load(String filename); §1.18.12
    public static void loadLibrary(String libname); §1.18.13
    public static void runFinalization(); §1.18.14
    public static void setProperties(Properties props); §1.18.15
    public static void setSecurityManager(SecurityManager s); §1.18.16
}
```

The System class contains a number of useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the System class are standard input, standard output, and error output streams, access to externally defined "properties", a means of loading files and libraries, and a utility method for quickly copying a portion of an array.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


System.err

public static `PrintStream` `err`

The "standard" output stream. This stream is already open and ready to accept output data.

Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable `out`, has been redirected to a file or other destination that is typically not continuously monitored.

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

System.in

public static InputStream in

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.out

`public static PrintStream out`

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

For simple standalone Java applications, a typical way to write a line of output data is:

```
System.out.println(data)
```

See the `println` methods ([§2.18.15](#)-[§2.18.24](#)) in class `PrintStream`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


System.arraycopy

`public static void`

```
arraycopy(Object src, int src_position, Object dst, int dst_position,  
int length)
```

A subsequence of array components are copied from the source array referenced by src to the destination array referenced by dst. The number of components copied is equal to the length argument. The components at positions srcOffset through srcOffset+length-1 in the source array are copied into positions dstOffset through dstOffset+length-1, respectively, of the destination array.

If the src and dst arguments refer to the same array object, then the copying is performed as if the components at positions srcOffset through srcOffset+length-1 were first copied to a temporary array with length components and then the contents of the temporary array were copied into positions dstOffset through dstOffset+length-1 of the argument array.

If any of the following is true, an ArrayStoreException is thrown and the destination is not modified:

- The src argument refers to an object that is not an array.
- The dst argument refers to an object that is not an array.
- The src argument and dst argument refer to arrays whose component types are different primitive types.
- The src argument refers to an array with a primitive component type and the dst argument refers to an array with a reference component type.
- The src argument refers to an array with a reference component type and the dst argument refers to an array with a primitive component type.

Otherwise, if any of the following is true, an ArrayIndexOutOfBoundsException is thrown and the destination is not modified:

- The srcOffset argument is negative.
- The dstOffset argument is negative.
- The length argument is negative.
- srcOffset+length is greater than src.length, the length of the source array.
- dstOffset+length is greater than dst.length, the length of the destination array.

Otherwise, if any actual component of the source array from position srcOffset through

srcOffset+length-1 cannot be converted to the component type of the destination array by assignment conversion, an `ArrayStoreException` is thrown. In this case, let k be the smallest nonnegative integer less than length such that src[srcOffset+k] cannot be converted to the component type of the destination array; when the exception is thrown, source array components from positions srcOffset through srcOffset+k-1 will already have been copied to destination array positions dstOffset through dstOffset+k-1 and no other positions of the destination array will have been modified.

Parameters:

src- the source array
srcpos- start position in the source array
dest- the destination array
destpos- start position in the destination data
length- the number of array elements to be copied

Throws

`ArrayIndexOutOfBoundsException` ([I-§1.25](#))
If copy would cause access of data outside array bounds.

Throws

`ArrayStoreException` ([I-§1.26](#))
If an element in the src array could not be stored into the dest array due to a type mismatch.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.currentTimeMillis

`public static long currentTimeMillis()`

Gets the current time.

See the description of the class `Date` ([I-§3.2](#)) for a discussion of slight discrepancies that may arise between "computer time" and UTC (Coordinated Universal Time).

Returns:

the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

System.exit

public static void exit(int status)

Terminates the currently running Java Virtual Machine. The argument serves as a status code; by convention, a nonzero status code indicates abnormal termination.

This method calls the exit method (I-§1.14.5) in class Runtime. This method never returns normally.

Parameters:

status- exit status

Throws

SecurityException (I-§1.43)

If the current thread cannot exit with the specified status.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.gc

public static void gc()

This method calls the gc method (I-§1.14.7) in class Runtime.

Calling this method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all unused objects.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.getProperties

public static Properties **getProperties()**

Determines the current system properties.

If there is a security manager, its checkPropertiesAccess method (I-§1.15.16) is called with no arguments. This may result in a security exception (I-§1.43).

The current set of system properties is returned as a Properties object (I-§3.6). If there is no current set of system properties, a set of system properties is first created and initialized.

This set of system properties always includes values for the following keys:

Key	Description of associated value
java.version	Java version number
java.vendor	Java-vendor-specific string
java.vendor.url	Java vendor URL
java.home	Java installation directory
java.class.version	Java class <u>format</u> version number
java.class.path	Java classpath
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
file.separator	File <u>separator</u> ("/" on Unix)
path.separator	Path <u>separator</u> (":" on Unix)
line.separator	Line <u>separator</u> ("\n" on Unix)
user.name	User account name
user.home	User home directory
user.dir	User's current working directory

Throws

SecurityException (I-§1.43)

If the current thread cannot access the system properties.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


System.getProperty

public static String getProperty(String key)

Gets the system property indicated by the specified key.

First, if there is a security manager, its checkPropertyAccess method (I-§1.15.17) is called with the key as its argument. This may result in a system exception (I-§1.43).

If there is no current set of system properties, a set of system properties is first created and initialized in the same manner as for the getProperty method (I-§1.18.8).

Parameters:

key– the name of the system property

Returns:

the string value of the system property, or null if there is no property with that key.

Throws

SecurityException (I-§1.43)

If the current thread cannot access the system properties or the specified property.

public static String getProperty(String key, String def)

Gets the system property indicated by the specified key.

First, if there is a security manager, its checkPropertyAccess method (I-§1.15.17) is called with the key as its argument.

If there is no current set of system properties, a set of system properties is first created and initialized in the same manner as for the getProperty method (I-§1.18.8).

Parameters:

key– the name of the system property

def– a default value

Returns:

the string value of the system property, or the default value if there is no property with that key.

Throws

SecurityException ([I-§1.43](#))

If the current thread cannot access the system properties or the specified property.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.getSecurityManager

public static SecurityManager getSecurityManager()

Returns:

if a security manager has already been established for the current application, then that security manager is returned; otherwise, null is returned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.load

public static void load(String filename)

Loads the given filename as a dynamic library. The filename argument must be a complete path name.

This method calls the load method (I-§1.14.11) in class Runtime.

Parameters:

filename- the file to load

Throws

UnsatisfiedLinkError (I-§1.61)

If the file does not exist.

Throws

SecurityException (I-§1.43)

If the current thread cannot load the specified dynamic library.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.loadLibrary

public static void loadLibrary(String libname)

Loads the system library specified by the libname argument. The manner in which a library name is mapped to the actual system library is system dependent.

Parameters:

libname- the name of the library

Throws

UnsatisfiedLinkError (I-§1.61)

If the library does not exist.

Throws

SecurityException (I-§1.43)

If the current thread cannot load the specified dynamic library.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.runFinalization

public static void runFinalization()

This method calls the runFinalization method (I-§1.14.13) in class Runtime.

Calling this method suggests that the Java Virtual Machine expend effort toward running the finalize methods of objects that have been found to be discarded but whose finalize methods have not yet been run. When control returns from the method call, the Java Virtual Machine has made a best effort to complete all outstanding finalizations.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.setProperties

`public static void setProperties(Properties props)`

Sets the system properties to the Properties argument.

First, if there is a security manager, its checkPropertiesAccess method (I-§1.15.16) is called with no arguments. This may result in a security exception (I-§1.43).

The argument becomes the current set of system properties for use by the getProperty method. If the argument is null, then the current set of system properties is forgotten.

Parameters:

`props`— the new system properties

Throws

SecurityException (I-§1.43)

If the current thread cannot set the system properties.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

System.setSecurityManager

public static void setSecurityManager(SecurityManager s)

If a security manager has already been established for the currently running Java application, a SecurityException is thrown. Otherwise, the argument is established as the current security manager. If the argument is null and no security manager has been established, then no action is taken and the method simply returns.

Parameters:

s – the security manager

Throws

SecurityException ([I-§1.43](#))

If the security manager has already been set.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.19 Class Thread

```
public class java.lang.Thread
    extends java.lang.Object (I-§1.12)
    implements java.lang.Runnable (I-§1.23)
{
    // Fields
    public final static int MAX_PRIORITY; §1.19.1
    public final static int MIN_PRIORITY; §1.19.2
    public final static int NORM_PRIORITY; §1.19.3

    // Constructors
    public Thread(); §1.19.4
    public Thread(Runnable target); §1.19.5
    public Thread(Runnable target, String name); §1.19.6
    public Thread(String name); §1.19.7
    public Thread(ThreadGroup group, Runnable target); §1.19.8
    public Thread(ThreadGroup group,
        Runnable target, String name); §1.19.9
    public Thread(ThreadGroup group, String name); §1.19.10

    // Methods
    public static int activeCount(); §1.19.11
    public void checkAccess(); §1.19.12
    public int countStackFrames(); §1.19.13
    public static Thread currentThread(); §1.19.14
    public void destroy(); §1.19.15
    public static void dumpStack(); §1.19.16
    public static int enumerate(Thread tarray[]); §1.19.17
    public final String getName(); §1.19.18
    public final int getPriority(); §1.19.19
    public final ThreadGroup getThreadGroup(); §1.19.20
    public void interrupt(); §1.19.21
    public static boolean interrupted(); §1.19.22
    public final boolean isAlive(); §1.19.23
    public final boolean isDaemon(); §1.19.24
    public boolean isInterrupted(); §1.19.25
    public final void join(); §1.19.26
    public final void join(long millis); §1.19.27
    public final void join(long millis, int nanos); §1.19.28
    public final void resume(); §1.19.29
    public void run(); §1.19.30
    public final void setDaemon(boolean on); §1.19.31
    public final void setName(String name); §1.19.32
    public final void setPriority(int newPriority); §1.19.33
```



```

    public static void sleep(long millis);           §1.19.34
    public static void sleep(long millis, int nanos) §1.19.35
    public void start();                             §1.19.36
    public final void stop();                         §1.19.37
    public final void stop(Throwable obj);           §1.19.38
    public final void suspend();                   §1.19.39
    public String toString();                       §1.19.40
    public static void yield();                     §1.19.41
}

```

A thread is a thread of executing in a program. The Java Virtual Machine allows an application to have multiple threads of executing concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not be marked as a dmon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a dmon thread if and only if the creating thread is a dmon.

When a Java Virtual Machine starts, there is usually a single non-dmon thread (which typically calls the method named main of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

- The exit method (I-§1.14.5) of class Runtime has been called and the security manager has permitted the exit operation to take place.
- All threads that are not dmon threads have died, either by returning from the call to the run method (I-§1.19.30) or by performing the stop method (I-§1.19.37).

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of Thread. This subclass should override the run method of class Thread. An instance of the subclass can then be allocated and started. For example, a thread whose job is to compute primes larger than a stated value could be written as follows:

```

class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
        ...
    }
}

```

The following code would then create a thread and start it running:

```

PrimeThread p = new PrimeThread(143);

```



```
p.start();
```

The other way to create a thread is to declare a class which implements the Runnable interface (I-§1.23). That class then implements the run method. An instance of the class can then be allocated, passed as an argument when creating Thread, and started. The same example in this other style looks like the following:

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
        ...
    }
}
```

The following code would then create a thread and start it running:

```
PrimeRun p = new PrimeRun(143);

new Thread(p).start();
```

Every thread has a name for identification purposes. More than one thread may have the same name. If a name is not specified when a thread is created, a new name is generated for it.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.MAX_PRIORITY

```
public final static int MAX_PRIORITY = 10
```

The maximum priority that a thread can have.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.MIN_PRIORITY

```
public final static int MIN_PRIORITY = 1
```

The minimum priority that a thread can have.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.NORM_PRIORITY

```
public final static int NORM_PRIORITY = 5
```

The default priority that is assigned to a thread.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.Thread

public Thread()

Allocates a new Thread object. This constructor has the same effect as Thread(null, null, gname) (I-§1.19.9) where gname is a newly generated name. Automatically generated names are of the form "Thread-"+n where n is an integer.

public Thread(Runnable target)

Allocates a new Thread object. This constructor has the same effect as Thread(null, target, gname) (I-§1.19.9) where gname is a newly generated name. Automatically generated names are of the form "Thread-"+n where n is an integer.

Parameters:

target- the object whose run method is called

public Thread(Runnable target, String name)

Allocates a new Thread object. This constructor has the same effect as Thread(null, target, name) (I-§1.19.9).

Parameters:

target- the object whose run method is called

name- the name of the new thread

public Thread(String name)

Allocates a new Thread object. This constructor has the same effect as Thread(null, null, name) (I-§1.19.9).

Parameters:

name- the name of the new thread

public Thread(ThreadGroup group, Runnable target)

Allocates a new Thread object. This constructor has the same effect as Thread(group, target, gname) (I-§1.19.9) where gname is a newly generated name. Automatically generated names are of the form "Thread-"+n where n is an integer.

Parameters:

group- the thread group

target- the object whose run method is called

Throws

SecurityException (I-§1.43)

If the current thread cannot create a thread in the specified thread group.

public Thread(ThreadGroup group, Runnable target,
String name)

Allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group.

If group is not null, the checkAccess method (I-§1.20.5) of that thread group is called with no arguments. This may result in throwing a SecurityException. If group is null, the new process belongs to the same group as the thread this is created the new thread.

If the target argument is not null, the run method of the target (I-§1.23.1) is called when this thread is started. If the target argument is null, this thread's run method (I-§1.19.30) is called when this thread is started.

The priority of the newly created thread is set equal to the priority of the thread creating it, that is, the currently running thread. The method setPriority (I-§1.19.33) may be used to change the priority to a new value.

The newly created thread is initially marked as being a dmon thread if the thread creating it is currently marked as a dmon thread. The method setDaemon (I-§1.19.31) may be used to change whether or not a thread is a dmon.

Parameters:

group- the thread group

target- the object whose run method is called

name- the name of the new thread

Throws

SecurityException (I-§1.43)

If the current thread cannot create a thread in the specified thread group.

public Thread(ThreadGroup group, String name)

Allocates a new Thread object. This constructor has the same effect as Thread(group, null, name) (I-§1.19.9)

Parameters:

group- the thread group

name- the name of the new thread

Throws

SecurityException (I-§1.43)

If the current thread cannot create a thread in the specified thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.activeCount

public static int activeCount()

Returns:

the current number of threads in this thread's thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.checkAccess

public void checkAccess()

Determines if the currently running thread has permission to modify this thread.

If there is a security manager, its checkAccess method (I-§1.15.4) is called with this thread as its argument. This may result in throwing a SecurityException.

Throws

SecurityException (I-§1.43)

If the current thread is not allowed to access this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.countStackFrames

public int countStackFrames()

Counts the number of stack frames in this thread. The thread must be suspended.

Returns:

the number of stack frames in this thread.

Throws

IllegalThreadStateException (I-§1.34)

If this thread is not suspended.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.currentThread

public static Thread currentThread()

Finds the currently executing thread.

Returns:

the currently executing thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.destroy1

public void destroy()

Destroys this thread, without any cleanup. Any monitors it has locked remain locked.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.dumpStack

`public static void dumpStack()`

Prints a stack trace of the current thread. Used only for debugging.

See Also:

printStackTrace in class Throwable ([I-§1.21.5](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.enumerate

public static int enumerate(Thread tarray[])

Copies into the array argument every thread in this thread's thread group. This method simply calls the enumerate method (I-§1.20.7) of this thread's thread group with the array argument.

Returns:

the number of threads put into the array.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.getName

```
public final String getName()
```

Returns:

this thread's name.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.getPriority

```
public final int getPriority()
```

Returns:

this thread's current priority.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.getThreadGroup

```
public final ThreadGroup getThreadGroup()
```

Returns:

this thread's thread group.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.interrupt

public void interrupt()

Interrupts this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.interrupted

public static boolean interrupted()

Returns:

true if the current thread has been interrupted; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.isAlive

`public final boolean isAlive()`

Determines if this thread is alive; if it has been started and has not yet died.

Returns:

true if this thread is alive; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.isDaemon

```
public final boolean isDaemon()
```

Returns:

true if this thread is a dmon thread; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.isInterrupted

public boolean isInterrupted()

Returns:

true if this thread has been interrupted; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.join

`public final void join()
throws InterruptedException`

Waits for this thread to die.

Throws

InterruptedException ([I-§1.37](#))

Another thread has interrupted the current thread.

`public final void join(long millis)
throws InterruptedException`

Waits at most `millis` milliseconds for for this thread to die. A timeout of 0 means to wait forever.

Parameters:

`millis`- the time to wait in milliseconds

Throws

InterruptedException ([I-§1.37](#))

Another thread has interrupted the current thread.

`public final void join(long millis, int nanos)
throws InterruptedException`

Waits at most `millis` milliseconds plus `nanos` nanoseconds for this thread to die.

Parameters:

`millis`- the time to wait in milliseconds

`nanos`- 0-999999 additional nanoseconds to wait

Throws

InterruptedException ([I-§1.37](#))

Another thread has interrupted the current thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.resume

public final void resume()

Resumes a suspended thread.

The checkAccess method (I-§1.19.12) of this thread is called with no arguments. This may result in throwing a SecurityException (in the current thread).

If the thread is alive (I-§1.19.23) but suspended, it is resumed and is permitted to make progress in its execution.

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.run

public void run()

If this thread was constructed using a separate Runnable run object (see I-§1.19.9), then that Runnable object's run method (I-§1.23.1) is called. Otherwise, this method does nothing and returns.

Subclasses of Thread should override this method.

See Also:

start (I-§1.19.36)
stop (I-§1.19.37).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.setDaemon

`public final void setDaemon(boolean on)`

Marks this thread as either a daemon thread or a user thread. The Java Virtual Machine exits when the only threads running are all daemon threads.

This method must be called before the thread is started.

Parameters:

`on` – if true, marks this thread as a daemon thread

Throws

IllegalThreadStateException ([I-§1.34](#))

If this thread is active.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.setName

```
public final void setName(String name)
```

Changes the name of this thread to be equal to the argument name.

First the checkAccess method (I-§1.19.12) of this thread is called with no arguments. This may result in throwing a SecurityException.

Parameters:

name- the new name for this thread

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Thread.setPriority

`public final void setPriority(int newPriority)`

Changes the priority of this thread.

First the checkAccess method (I-§1.19.12) of this thread is called with no arguments. This may result in throwing a SecurityException.

Otherwise, the priority of this thread is set to the smaller of the specified newPriority and the maximum permitted priority (I-§1.20.11) of the thread's thread group (I-§1.19.20).

Throws

IllegalArgumentException (I-§1.32)

If the priority is not in the range MIN_PRIORITY to MAX_PRIORITY.

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.sleep

public static void sleep(long millis)
throws InterruptedException

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors (see I-§1.12.7 for further discussion of monitors).

Parameters:

`millis`– the length of time to sleep in milliseconds

Throws

InterruptedException (I-§1.37)

Another thread has interrupted this thread.

public static void sleep(long millis, int nanos)
throws InterruptedException

Causes the currently executing thread to sleep (cease execution) for the specified number of milliseconds plus the specified number of nanoseconds. The thread does not lose ownership of any monitors (see I-§1.12.7 for further discussion of monitors).

Parameters:

`millis`– the length of time to sleep in milliseconds

`nanos`– 0-999999 additional nanoseconds to sleep.

Throws

InterruptedException (I-§1.37)

Another thread has interrupted this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.start

public void start()

Causes this thread to begin execution; the Java Virtual Machine calls the run method (I-§1.19.30) of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the start method) and the other thread (which executes its run method).

Throws

IllegalThreadStateException (I-§1.34)

If the thread was already started.

See Also:

run (I-§1.19.30)

stop (I-§1.19.37).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.stop

`public final void stop()`

Forces this thread to stop executing.

The checkAccess method ([I-§1.19.12](#)) of this thread is called with no arguments. This may result in throwing a SecurityException (in the current thread).

The thread represented by the this thread is forced to stop whatever it is doing abnormally and to throw a newly created ThreadDeath ([I-§1.59](#)) object as an exception.

It is permitted to stop a thread that has not yet been started. If the thread is eventually started, it immediately terminates.

An application should not normally try to catch ThreadDeath unless it must do some extraordinary cleanup operation (note that the throwing of ThreadDeath will cause finally clauses of try statements to be executed before the thread officially dies). If a catch clause does catch a ThreadDeath object, it is important to rethrow the object so that the thread actually dies.

The top-level error handler that reacts to otherwise uncaught exceptions ([I-§1.20.23](#)) does not print out a message or otherwise notify the application if the uncaught exception is an instance of ThreadDeath.

See Also:

start ([I-§1.19.36](#))
run ([I-§1.19.30](#)).

Throws

SecurityException ([I-§1.43](#))

If the current thread cannot modify this thread.

`public final void stop(Throwable obj)`

Forces this thread to stop executing.

The checkAccess method (I-§1.19.12) of this thread is called with no arguments. This may result in throwing a SecurityException (in the current thread).

If the argument obj is null, a NullPointerException is thrown (in the current thread).

The thread represented by this thread is forced to complete whatever it is doing abnormally and to throw the Throwable object obj as an exception. This is an unusual action to take; normally, the stop method that takes no arguments (I-§1.19.37) should be used.

It is permitted to stop a thread that has not yet been started. If the thread is eventually started, it immediately terminates.

Parameters:

obj – the Throwable object to be thrown

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.suspend

public final void suspend()

Suspends this thread.

The checkAccess method (I-§1.19.12) of this thread is called with no arguments. This may result in throwing a SecurityException (in the current thread).

If the thread is alive (I-§1.19.23), it is suspended and makes no further progress unless it is resumed.

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.toString

public String toString()

Returns:

a string representation of this thread.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Thread.yield

public static void yield()

Causes the currently executing thread object to temporarily pause and allow other threads to execute.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹Unimplemented in Java 1.1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.20 Class ThreadGroup

```
public class java.lang.ThreadGroup
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public ThreadGroup(String name); §1.20.1
    public ThreadGroup(ThreadGroup parent, String name); §1.20.2

    // Methods
    public int activeCount(); §1.20.3
    public int activeGroupCount(); §1.20.4
    public final void checkAccess(); §1.20.5
    public final void destroy(); §1.20.6
    public int enumerate(Thread list[]); §1.20.7
    public int enumerate(Thread list[], boolean recurse); §1.20.8
    public int enumerate(ThreadGroup list[]); §1.20.9
    public int enumerate(ThreadGroup list[], boolean recurse); §1.20.10
    public final int getMaxPriority(); §1.20.11
    public final String getName(); §1.20.12
    public final ThreadGroup getParent(); §1.20.13
    public final boolean isDaemon(); §1.20.14
    public void list(); §1.20.15
    public final boolean parentOf(ThreadGroup g); §1.20.16
    public final void resume(); §1.20.17
    public final void setDaemon(boolean daemon); §1.20.18
    public final void setMaxPriority(int pri); §1.20.19
    public final void stop(); §1.20.20
    public final void suspend(); §1.20.21
    public String toString(); §1.20.22
    public void uncaughtException(Thread t, Throwable e); §1.20.23
}
```

A thread group represents a set of threads. In addition, a thread group can also include other thread groups. The thread groups form a tree in which every thread group except the initial thread group has a parent.

A thread is allowed to access information about its own thread group, but not to access information about its thread group's parent thread group.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.ThreadGroup

public ThreadGroup(String name)

Constructs a new thread group, whose parent is the thread group of the currently running thread.

Parameters:

name- the name of the new thread group

public ThreadGroup(ThreadGroup parent, String name)

Creates a new thread group whose parent is the specified thread group.

The checkAccess method (I-§1.20.5) of the parent thread group is called with no arguments; this may result in a security exception(I-§1.43).

Parameters:

parent- the parent thread group

name- the name of the new thread group

Throws

NullPointerException (I-§1.40)

If the thread group argument is null.

Throws

SecurityException (I-§1.43)

If the current thread cannot create a thread in the specified thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.activeCount

public int activeCount()

Returns:

The number of active threads in this thread group and in any other thread group that has this thread group as an ancestor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.activeGroupCount

```
public int activeGroupCount()
```

Returns:

The number of active thread groups with this thread group as an ancestor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.checkAccess

`public final void checkAccess()`

Determines if the currently running thread has permission to modify this thread group.

If there is a security manager, its checkAccess method (I-§1.15.5) is called with this thread group as its argument. This may result in throwing a SecurityException.

Throws

SecurityException (I-§1.43)

If the current thread is not allowed to access this thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.destroy

public final void destroy()

Destroys this thread group and all of its sub-groups. This thread group must be empty, indicating that all threads that had been in this thread group have since stopped.

Throws

IllegalThreadStateException (I-§1.34)

If the thread group is not empty or if the thread group has already destroyed.

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.enumerate

```
public int enumerate(Thread list[])
```

Copies into the specified array every active thread in this thread group and its sub-groups.

An application should use the activeCount method ([I-§1.20.3](#)) to get an estimate of how big the array should be. If the array is too short to hold all the threads, the extra threads are silently ignored.

Parameters:

`list`- an array into which to place the list of threads

Returns:

the number of threads put into the array.

```
public int enumerate(Thread list[], boolean recurse)
```

Copies into the specified array every active thread in this thread group. If the recurse flag is true references to every active thread in this thread's sub-groups are also included. If the array is too short to hold all the threads, the extra threads are silently ignored.

An application should use the activeCount method ([I-§1.20.3](#)) to get an estimate of how big the array should be.

Parameters:

`list`- an array into which to place the list of threads

`recurse`- flag indicating whether to also include threads in sub-thread groups

Returns:

the number of threads placed into the array.

```
public int enumerate(ThreadGroup list[])
```

Copies into the specified array references to every active sub-group in this thread group.

An application should use the `activeGroupCount` [method \(I-§1.20.4\)](#) to get an estimate of how big the array should be. If the array is too short to hold all the thread groups, the extra thread groups are silently ignored.

Parameters:

`list`- an array into which to place the list of thread groups

Returns:

the number of thread groups put into the array.

```
public int enumerate(ThreadGroup list[], boolean recurse)
```

Copies into the specified array references to every active sub-group in this thread group. If the recurse flag is true references to every active, all sub-thread groups of the sub-thread groups and so forth are also included.

An application should use the `activeGroupCount` [method \(I-§1.20.4\)](#) to get an estimate of how big the array should be.

Parameters:

`list`- an array into which to place the list of threads

`recurse`- flag indicating whether to recursively enumerate all included thread groups

Returns:

the number of thread groups put into the array.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.getMaxPriority

```
public final int getMaxPriority()
```

Returns:

the maximum priority that a thread in this thread group can have.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.getName

```
public final String getName()
```

Returns:

the name of this thread group.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.getParent

```
public final ThreadGroup getParent()
```

Returns:

the parent of this thread group. The top-level thread group is the only thread group whose parent is null.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadGroup.isDaemon

public final boolean isDaemon()

Determines if this thread group is a dmon thread group. A dmon thread group is automatically destroyed when its last thread is stopped or its last thread group is destroyed.

Returns:

true if this thread group is a dmon thread group; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.list

public void list()

Prints information about this thread group to the standard output. This method is useful only for debugging.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.parentOf

`public final boolean parentOf(ThreadGroup g)`

Determines if this thread group is either the thread group argument or one of its ancestor thread groups.

Parameters:

g- a thread group

Returns:

true if this thread group is the thread group argument or one of its ancestor thread groups;
false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.resume

`public final void resume()`

Resumes all processes in this thread group.

First, the checkAccess method ([I-§1.20.5](#)) of this thread group is called with no arguments; this may result in a security exception ([I-§1.43](#)).

This method then calls the resume method ([I-§1.19.29](#)) on all the threads in this thread group and in all of its sub-thread groups.

Throws

SecurityException ([I-§1.43](#))

If the current thread is not allowed to access this thread group or any of the threads in the thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.setDaemon

public **final** **void** **setDaemon**(**boolean** **daemon**)

Sets whether this thread group is a dmon thread group.

The checkAccess method (I-§1.20.5) of this thread group is called with no arguments; this may result in a security exception (I-§1.43).

Marks whether this thread is a dmon thread group. A dmon thread group is automatically destroyed when its last thread is stopped or its last thread group is destroyed.

Parameters:

daemon– if true, marks this thread group as a dmon thread group; otherwise, marks this thread group as normal

Throws

SecurityException (I-§1.43)

If the current thread cannot modify this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.setMaxPriority

public **final** **void** **setMaxPriority**(**int** **pri**)

Sets the maximum priority of the group.

The `checkAccess` [method \(I-§1.20.5\)](#) of this [thread](#) group is called with no arguments; this may result in a security [exception \(I-§1.43\)](#).

Threads in the [thread](#) group that already have a higher priority are not affected.

Parameters:

`pri` - the new priority of the Thread group

Throws

SecurityException [\(I-§1.43\)](#)

If the current [thread](#) cannot modify this [thread](#) group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.stop

`public final void stop()`

Stops all processes in this thread group.

The checkAccess method ([I-§1.20.5](#)) of this thread group is called with no arguments; this may result in a security exception ([I-§1.43](#)).

This method then calls the stop method ([I-§1.19.37](#)) on all the threads in this thread group and in all of its sub-thread groups.

Throws

SecurityException ([I-§1.43](#))

If the current thread is not allowed to access this thread group or any of the threads in the thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.suspend

`public final void suspend()`

Suspends all processes in this thread group.

The checkAccess method (I-§1.20.5) of this thread group is called with no arguments; this may result in a security exception(I-§1.43).

This method then calls the suspend method (I-§1.19.39) on all the threads in this thread group and in all of its sub-thread groups.

Throws

SecurityException (I-§1.43)

If the current thread is not allowed to access this thread group or any of the threads in the thread group.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.toString

public String toString()

Returns:

a string representation of this thread group.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ThreadGroup.uncaughtException

public void uncaughtException(Thread t, Throwable e)

The Java Virtual Machine calls this method when a thread in this thread group stops because of an uncaught exception.

The uncaughtException method of ThreadGroup does the following:

If this thread group has a parent thread group, the uncaughtException method of that parent is called with the same two arguments.

Otherwise, this method determines if the Throwable argument is an instance of ThreadDeath (I-§1.59). If so, nothing special is done. Otherwise, the Throwable's printStackTrace method (I-§1.21.6) is called to print a stack back trace to the standard error stream (I-§1.18.1).

Applications can override this method in subclasses of ThreadGroup to provide alternative handling of uncaught exceptions.

Parameters:

- t– the thread that is about to exit
- e– the uncaught exception

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.21 Class Throwable

```
public class java.lang.Throwable
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public Throwable(); §1.21.1
    public Throwable(String message); §1.21.2

    // Methods
    public Throwable fillInStackTrace(); §1.21.3
    public String getMessage(); §1.21.4
    public void printStackTrace(); §1.21.5
    public void printStackTrace(PrintStream s); §1.21.6
    public String toString(); §1.21.7
}
```

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause.

A Throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error.

The following is one example of catching an exception:

```
try {
    int a[] = new int[2];
    a[4];
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("exception: " + e.getMessage());
    e.printStackTrace();
}
```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Throwable.Throwable

public **Throwable** ()

Constructs a new Throwable with no detail message. The stack trace is automatically filled in.

public **Throwable** (String message)

Constructs a new Throwable with the specified detail message. The stack trace is automatically filled in.

Parameters:

message– the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Throwable.fillInStackTrace

public **Throwable** **fillInStackTrace()**

Fills in the execution stack trace. This method is useful when an application is re-throwing an error or exception. For example:

```
try {  
    a = b / c;  
} catch(ArithmeticThrowable e) {  
    a = Number.MAX_VALUE;  
    throw e.fillInStackTrace();  
}  
Returns:
```

this Throwable object.

See Also:

printStackTrace (I-§1.21.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Throwable.getMessage

public String getMessage()

Returns:

the detail message of this Throwable, or null if this Throwable does not have a detailed message.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Throwable.printStackTrace

public void printStackTrace()

Prints this Throwable and its backtrace to the standard error (I-§1.18.1) output stream.

public void printStackTrace(PrintStream s)

Prints this Throwable and its backtrace to the specified print stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Throwable.toString

public String toString()

Returns:

a string representation of this Throwable.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.22 Interface Cloneable

```
public interface java.lang.Cloneable  
{  
}
```

A class implements the Cloneable interface to indicate to the clone method (I-§1.12.2) in class Object that it is legal for that method to make a field-for-field copy of instances of that class.

Attempts to clone instances that do not implement the Cloneable interface result in the exception CloneNotSupportedException (I-§1.29) being thrown.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.23 Interface Runnable

```
public interface java.lang.Runnable
{
    // Methods
    public abstract void run();    §1.23.1
}
```

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread (I-§1.19). The class must define a method of no arguments called run.

See Also:

Thread (I-§1.19).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Runnable.run

public abstract void run()

When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

See Also:

run in class Thread (I-§1.19.30).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.24 Class ArithmeticException

```
public class java.lang.ArithmeticException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public ArithmeticException(); §1.24.1
    public ArithmeticException(String s); §1.24.2
}
```

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ArithmeticException.ArithmeticException

public **ArithmeticException**()

Constructs an ArithmeticException with no detail message.

public **ArithmeticException**(String s)

Constructs an ArithmeticException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.25 Class ArrayIndexOutOfBoundsException

```
public class java.lang.ArrayIndexOutOfBoundsException
    extends java.lang.IndexOutOfBoundsException (I-§1.35)
{
    // Constructors
    public ArrayIndexOutOfBoundsException(); §1.25.1
    public ArrayIndexOutOfBoundsException(int index); §1.25.2
    public ArrayIndexOutOfBoundsException(String s); §1.25.3
}
```

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ArrayIndexOutOfBoundsException.ArrayIndexOutOfBoundsException

public `ArrayIndexOutOfBoundsException()`

Constructs an `ArrayIndexOutOfBoundsException` class with no detail message.

public `ArrayIndexOutOfBoundsException(int index)`

Constructs a new `ArrayIndexOutOfBoundsException` class with an argument indicating the illegal index.

Parameters:

`index`– the illegal index

public `ArrayIndexOutOfBoundsException(String s)`

Constructs an `ArrayIndexOutOfBoundsException` class with the specified detail message.

Parameters:

`s`– the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.26 Class ArrayStoreException

```
public class java.lang.ArrayStoreException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public ArrayStoreException(); §1.26.1
    public ArrayStoreException(String s); §1.26.2
}
```

Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an ArrayStoreException:

```
Object x[] = new String[3];
```

```
x[0] = new Integer(0);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ArrayStoreException.ArrayStoreException

public **ArrayStoreException**()

Constructs a **ArrayStoreException** with no detail message.

public **ArrayStoreException**(String s)

Constructs a **ArrayStoreException** with the specified detail message.

Parameters:

s – the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.27 Class ClassCastException

```
public class java.lang.ClassCastException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public ClassCastException(); §1.27.1
    public ClassCastException(String s); §1.27.2
}
```

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a ClassCastException:

```
Object x = new Integer(0);
System.out.println((String)x);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ClassCastException.ClassCastException

public `ClassCastException()`

Constructs a ClassCastException with no detail message.

public `ClassCastException(String s)`

Constructs a ClassCastException with the specified detail message.

Parameters:

`s` – the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.28 Class ClassNotFoundException

```
public class java.lang.ClassNotFoundException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public ClassNotFoundException(); §1.28.1
    public ClassNotFoundException(String s); §1.28.2
}
```

Thrown when an application tries to load in a class through its string name using:

- the forName method (I-§1.3.1) in class `Class`
- the findSystemClass method (I-§1.4.3) in class `ClassLoader`
- the loadClass method in class `ClassLoader` (I-§1.4.4)

but no definition for the class with the specified name could be found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ClassNotFoundException.ClassNotFoundException

public `ClassNotFoundException()`

Constructs a `ClassNotFoundException` with no detail message.

public `ClassNotFoundException(String s)`

Constructs a `ClassNotFoundException` with the specified detail message.

Parameters:

`s` – the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.29 Class CloneNotSupportedException

```
public class java.lang.CloneNotSupportedException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public CloneNotSupportedException(); §1.29.1
    public CloneNotSupportedException(String s); §1.29.2
}
```

Thrown to indicate that the clone method (I-§1.12.2) in class Object has been called to clone an object, but that the object's class does not implement the Cloneable interface (I-§1.22).

Applications that override the clone method can also throw this exception to indicate that an object could not or should not be cloned.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CloneNotSupportedException.CloneNotSupportedException

public CloneNotSupportedException()

Constructs an CloneNotSupportedException with no detail message.

public CloneNotSupportedException(String s)

Constructs an CloneNotSupportedException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.30 Class Exception

```
public class java.lang.Exception
    extends java.lang.Throwable (I-§1.21)
{
    // Constructors
    public Exception(); §1.30.1
    public Exception(String s); §1.30.2
}
```

The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

See Also:

Error (I-§1.48).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Exception.Exception

public **Exception**()

Constructs an Exception with no specified detail message.

public **Exception**(String s)

Constructs a Exception with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.31 Class `IllegalAccessException`

```
public class java.lang.IllegalAccessException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public IllegalAccessException(); §1.31.1
    public IllegalAccessException(String s); §1.31.2
}
```

Thrown when an application tries to load in a class through its string name using

- the `forName` method (I-§1.3.1) in class `Class`
- the `findSystemClass` method (I-§1.4.3) in class `ClassLoader`
- the `loadClass` method in class `ClassLoader` (I-§1.4.4)

but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.

An instance of this class can also be thrown when an application tries to create an instance of a class using the `newInstance` method(I-§1.3.7) in class `Class`, but the current method does not have access to the appropriate zero-argument constructor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IllegalAccessException.IllegalAccessException

public `IllegalAccessException()`

Constructs a `IllegalAccessException` without a detail message.

public `IllegalAccessException(String s)`

Constructs a `IllegalAccessException` with a detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.32 Class IllegalArgumentException

```
public class java.lang.IllegalArgumentException
    extends java.lang.RuntimeException (l-§1.42)
{
    // Constructors
    public IllegalArgumentException(); §1.32.1
    public IllegalArgumentException(String s); §1.32.2
}
```

Thrown to indicate that a method has been passed an illegal or inappropriate argument.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IllegalArgumentException.IllegalArgumentException

public `IllegalArgumentException()`

Constructs an IllegalArgumentException with no detail message.

public `IllegalArgumentException(String s)`

Constructs an IllegalArgumentException with the specified detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.33 Class IllegalMonitorStateException

```
public class java.lang.IllegalMonitorStateException
    extends java.lang.RuntimeException (l-§1.42)
{
    // Constructors
    public IllegalMonitorStateException(); §1.33.1
    public IllegalMonitorStateException(String s); §1.33.2
}
```

Thrown to indicate that a thread has attempted to wait on an object's monitor (l-§1.12.10-§1.12.12) or to notify other threads waiting on an object's monitor (l-§1.12.7, §1.12.8) without owning the specified monitor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IllegalMonitorStateException.IllegalMonitorStateException

public IllegalMonitorStateException()

Constructs an IllegalMonitorStateException with no detail message.

public IllegalMonitorStateException(String s)

Constructs an IllegalMonitorStateException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.34 Class `IllegalThreadStateException`

```
public class java.lang.IllegalThreadStateException
    extends java.lang.IllegalArgumentException (I-§1.32)
{
    // Constructors
    public IllegalThreadStateException(); §1.34.1
    public IllegalThreadStateException(String s); §1.34.2
}
```

Thrown to indicate that a thread is not in an appropriate state for the requested operation. See, for example, the suspend (I-§1.19.39) and resume (I-§1.19.29) methods in class `Thread`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IllegalThreadStateException.IllegalThreadStateException

public `IllegalThreadStateException()`

Constructs an `IllegalThreadStateException` with no detail message.

public `IllegalThreadStateException(String s)`

Constructs an `IllegalThreadStateException` with the specified detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.35 Class IndexOutOfBoundsException

```
public class java.lang.IndexOutOfBoundsException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public IndexOutOfBoundsException(); §1.35.1
    public IndexOutOfBoundsException(String s); §1.35.2
}
```

Instances of this class are thrown indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

Applications can subclass this class to indicate similar exceptions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IndexOutOfRangeException.IndexOutOfRangeException

public IndexOutOfRangeException()

Constructs an IndexOutOfRangeException with no detail message.

public IndexOutOfRangeException(String s)

Constructs a IndexOutOfRangeException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.36 Class InstantiationException

```
public class java.lang.InstantiationException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public InstantiationException(); §1.36.1
    public InstantiationException(String s); §1.36.2
}
```

Thrown when an application tries to create an instance of a class using the newInstance method (I-§1.3.7) in class Class, but the specified class object cannot be instantiated because it is an interface or is an abstract class.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InstantiationExceptionInstantiationException

public `InstantiationException()`

Constructs an `InstantiationException` with no detail message.

public `InstantiationException(String s)`

Constructs an `InstantiationException` with the specified detail message.

Parameters:

`s` – the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.37 Class InterruptedException

```
public class java.lang.InterruptedException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public InterruptedException(); §1.37.1
    public InterruptedException(String s); §1.37.2
}
```

Thrown when a thread is waiting (I-§1.12.10-§1.12.12), sleeping (I-§1.19.34), or otherwise paused for a long time and another thread interrupts it using the interrupt method (I-§1.19.21) in class Thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InterruptedException. InterruptedException

public InterruptedException()

Constructs an InterruptedException with no detail message.

public InterruptedException(String s)

Constructs an InterruptedException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.38 Class NegativeArraySizeException

```
public class java.lang.NegativeArraySizeException  
    extends java.lang.RuntimeException (l-§1.42)  
{  
    // Constructors  
    public NegativeArraySizeException(); §1.38.1  
    public NegativeArraySizeException(String s); §1.38.2  
}
```

Thrown if an application tries to create an array with negative size.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NegativeArraySizeException.NegativeArraySizeException

public NegativeArraySizeException()

Constructs a NegativeArraySizeException with no detail message.

public NegativeArraySizeException(String s)

Constructs a NegativeArraySizeException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.39 Class NoSuchMethodException

```
public class java.lang.NoSuchMethodException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public NoSuchMethodException(); §1.39.1
    public NoSuchMethodException(String s); §1.39.2
}
```

This exception is obsolete.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NoSuchMethodException.NoSuchMethodException

public `NoSuchMethodException()`

Constructs a `NoSuchMethodException` without a detail message.

public `NoSuchMethodException(String s)`

Constructs a `NoSuchMethodException` with a detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.40 Class NullPointerException

```
public class java.lang.NullPointerException
    extends java.lang.RuntimeException (l-§1.42)
{
    // Constructors
    public NullPointerException(); §1.40.1
    public NullPointerException(String s); §1.40.2
}
```

Thrown when an application attempts to use null in a case where an object is required. These include:

- Calling the instance method of a null object.
- Accessing or modifying the field of the null object.
- Taking the length of null as if it were an array.
- Accessing or modifying the slots of null as if it were an array.
- Throwing null as if it were a Throwable value.

Applications should throw instances of this class to indicate other illegal uses of the null object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NullPointerException.NullPointerException

public `NullPointerException()`

Constructs a `NullPointerException` with no detail message.

public `NullPointerException(String s)`

Constructs a `NullPointerException` with the specified detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.41 Class NumberFormatException

```
public class java.lang.NumberFormatException
    extends java.lang.IllegalArgumentException (l-§1.32)
{
    // Constructors
    public NumberFormatException(); §1.41.1
    public NumberFormatException(String s); §1.41.2
}
```

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

NumberFormatException.NumberFormatException

public `NumberFormatException()`

Constructs a `NumberFormatException` with no detail message.

public `NumberFormatException(String s)`

Constructs a `NumberFormatException` with the specified detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.42 Class RuntimeException

```
public class java.lang.RuntimeException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public RuntimeException(); §1.42.1
    public RuntimeException(String s); §1.42.2
}
```

RuntimeException is the superclass of those exceptions which can be thrown during the normal operation of the Java Virtual Machine.

A method is not required to declare in its throws clause any subclasses of RuntimeException that might be thrown during the execution of the method but not caught.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RuntimeException RuntimeException

public RuntimeException()

Constructs a RuntimeException with no detail message.

public RuntimeException(String s)

Constructs a RuntimeException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.43 Class `SecurityException`

```
public class java.lang.SecurityException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public SecurityException(); §1.43.1
    public SecurityException(String s); §1.43.2
}
```

Thrown by the security manager (I-§1.15) to indicate a security violation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SecurityException.SecurityException

public SecurityException()

Constructs a SecurityException with no detail message.

public SecurityException(String s)

Constructs a SecurityException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.44 Class StringIndexOutOfBoundsException

```
public class java.lang.StringIndexOutOfBoundsException
    extends java.lang.IndexOutOfBoundsException (I-§1.35)
{
    // Constructors
    public StringIndexOutOfBoundsException(); §1.44.1
    public StringIndexOutOfBoundsException(int index); §1.44.2
    public StringIndexOutOfBoundsException(String s) §1.44.3
}
```

Thrown by the charAt method (I-§1.16.8) in class classString and by other String methods to indicate that an index is either negative or greater than or equal to the size of the string.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringIndexOutOfBoundsException.StringIndexOutOfBoundsException

public StringIndexOutOfBoundsException()

Constructs a StringIndexOutOfBoundsException with no detail message.

public StringIndexOutOfBoundsException(int index)

Constructs a new StringIndexOutOfBoundsException class with an argument indicating the illegal index.

Parameters:

index- the illegal index

public StringIndexOutOfBoundsException(String s)

Constructs a StringIndexOutOfBoundsException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.45 Class AbstractMethodError

```
public class java.lang.AbstractMethodError
    extends java.lang.IncompatibleClassChangeError (I-§1.50)
{
    // Constructors
    public AbstractMethodError(); §1.45.1
    public AbstractMethodError(String s); §1.45.2
}
```

Thrown when an application tries to call an abstract method. Normally, this error is caught by the compiler; this error can only occur at runtime if the definition of some class has incompatibly changed since the currently executing method was last compiled.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AbstractMethodError.AbstractMethodError

public AbstractMethodError ()

Constructs an AbstractMethodError with no detail message.

public AbstractMethodError (String s)

Constructs an AbstractMethodError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.46 Class ClassCircularityError

```
public class java.lang.ClassCircularityError
    extends java.lang.LinkageError (l-§1.53)
{
    // Constructors
    public ClassCircularityError(); §1.46.1
    public ClassCircularityError(String s); §1.46.2
}
```

This error is obsolete.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ClassCircularityError.ClassCircularityError

public ClassCircularityError()

Constructs a ClassCircularityError with no detail message.

public ClassCircularityError(String s)

Constructs a ClassCircularityError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.47 Class ClassFormatError

```
public class java.lang.ClassFormatError
    extends java.lang.LinkageError (l-§1.53)
{
    // Constructors
    public ClassFormatError(); §1.47.1
    public ClassFormatError(String s); §1.47.2
}
```

Thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ClassFormatError.ClassFormatError

public `ClassFormatError()`

Constructs a ClassFormatError with no detail message.

public `ClassFormatError(String s)`

Constructs a ClassFormatError with the specified detail message.

Parameters:

`s` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.48 Class Error

```
public class java.lang.Error
    extends java.lang.Throwable (I-§1.21)
{
    // Constructors
    public Error(); §1.48.1
    public Error(String s); §1.48.2
}
```

An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions. The ThreadDeath error (I-§1.59), though a "normal" condition, is also a subclass of Error since most applications should not try to catch it.

A method is not required to declare in its throws clause any subclasses of Error that might be thrown during the execution of the method but not caught, since these errors are abnormal conditions that should never occur.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Error.Error

public Error()

Constructs an Error with no specified detail message.

public Error(String s)

Constructs an Error with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.49 Class `IllegalAccessError`

```
public class java.lang.IllegalAccessError  
    extends java.lang.IncompatibleClassChangeError (I-§1.50)  
{  
    // Constructors  
    public IllegalAccessError(); §1.49.1  
    public IllegalAccessError(String s); §1.49.2  
}
```

Thrown if an application attempts to access or modify a field, or to call a method that it does not have access to.

Normally, this error is caught by the compiler; this error can only occur at run-time if the definition of a class has incompatibly changed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IllegalAccessError.IllegalAccessError

public IllegalAccessError()

Constructs an IllegalAccessError with no detail message.

public IllegalAccessError(String s)

Constructs an IllegalAccessError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.50 Class IncompatibleClassChangeError

```
public class java.lang.IncompatibleClassChangeError
    extends java.lang.LinkageError (l-§1.53)
{
    // Constructors
    public IncompatibleClassChangeError(); §1.50.1
    public IncompatibleClassChangeError(String s); §1.50.2
}
```

An instance of a subclass of IncompatibleClassChangeError is thrown to indicate that an incompatible class change has occurred to some class definition. The definition of some class, upon which the currently executing method depends, has since changed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IncompatibleClassChangeError.IncompatibleClassChangeError

public IncompatibleClassChangeError()

Constructs an IncompatibleClassChangeError with no detail message.

public IncompatibleClassChangeError(String s)

Constructs an IncompatibleClassChangeError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.51 Class InstantiationException

```
public class java.lang.InstantiationException
    extends java.lang.IncompatibleClassChangeError (I-§1.50)
{
    // Constructors
    public InstantiationException(); §1.51.1
    public InstantiationException(String s); §1.51.2
}
```

Thrown when an application tries to use the Java new construct to instantiate an abstract class or an interface.

Normally, this error is caught by the compiler; this error can only occur at run-time if the definition of a class has incompatibly changed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InstantiationError.InstantiationError

public InstantiationError()

Constructs an InstantiationError with no detail message.

public InstantiationError(String s)

Constructs an InstantiationError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.52 Class InternalError

```
public class java.lang.InternalError
    extends java.lang.VirtualMachineError (I-§1.63)
{
    // Constructors
    public InternalError(); §1.52.1
    public InternalError(String s); §1.52.2
}
```

Thrown to indicate some unexpected internal error has occurred in the Java Virtual Machine.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InternalError.InternalError

public InternalError()

Constructs an InternalError with no detail message.

public InternalError(String s)

Constructs an InternalError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.53 Class LinkageError

```
public class java.lang.LinkageError
    extends java.lang.Error (I-§1.48)
{
    // Constructors
    public LinkageError(); §1.53.1
    public LinkageError(String s); §1.53.2
}
```

Subclasses of LinkageError indicate that a class has some dependency on another class; however the latter class has incompatibly changed after the compilation of the former class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


LinkageError.LinkageError

public LinkageError()

Constructs a LinkageError with no detail message.

public LinkageError(String s)

Constructs a LinkageError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.54 Class NoClassDefFoundError

```
public class java.lang.NoClassDefFoundError
    extends java.lang.LinkageError (l-§1.53)
{
    // Constructors
    public NoClassDefFoundError(); §1.54.1
    public NoClassDefFoundError(String s); §1.54.2
}
```

Thrown if the Java Virtual Machine or a classloader tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the new expression) and no definition of the class could be found.

The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

NoClassDefFoundError.NoClassDefFoundError

public NoClassDefFoundError ()

Constructs a NoClassDefFoundError with no detail message.

public NoClassDefFoundError (String s)

Constructs a NoClassDefFoundError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.55 Class NoSuchFieldError

```
public class java.lang.NoSuchFieldError  
    extends java.lang.IncompatibleClassChangeError (I-§1.50)  
{  
    // Constructors  
    public NoSuchFieldError(); §1.55.1  
    public NoSuchFieldError(String s); §1.55.2  
}
```

Thrown if an application tries to access or modify a specified field of an object, and that object no longer has that field.

Normally, this error is caught by the compiler; this error can only occur at run-time if the definition of a class has incompatibly changed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NoSuchFieldError.NoSuchFieldError

public NoSuchFieldError()

Constructs a NoSuchFieldException with no detail message.

public NoSuchFieldError(String s)

Constructs a NoSuchFieldException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.56 Class NoSuchMethodError

```
public class java.lang.NoSuchMethodError  
    extends java.lang.IncompatibleClassChangeError (I-§1.50)  
{  
    // Constructors  
    public NoSuchMethodError(); §1.56.1  
    public NoSuchMethodError(String s); §1.56.2  
}
```

Thrown if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method.

Normally, this error is caught by the compiler; this error can only occur at run-time if the definition of a class has incompatibly changed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NoSuchMethodError.NoSuchMethodError

public NoSuchMethodError ()

Constructs a NoSuchMethodException with no detail message.

public NoSuchMethodError (String s)

Constructs a NoSuchMethodException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.57 Class OutOfMemoryError

```
public class java.lang.OutOfMemoryError  
    extends java.lang.VirtualMachineError (I-§1.63)  
{  
    // Constructors  
    public OutOfMemoryError(); §1.57.1  
    public OutOfMemoryError(String s); §1.57.2  
}
```

Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

OutOfMemoryError.OutOfMemoryError

public OutOfMemoryError()

Constructs an OutOfMemoryError with no detail message.

public OutOfMemoryError(String s)

Constructs an OutOfMemoryError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.58 Class StackOverflowError

```
public class java.lang.StackOverflowError
    extends java.lang.VirtualMachineError (I-§1.63)
{
    // Constructors
    public StackOverflowError(); §1.58.1
    public StackOverflowError(String s); §1.58.2
}
```

Thrown when a stack overflow occurs because an application recurses too deeply.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StackOverflowError.StackOverflowError

public StackOverflowError()

Constructs a StackOverflowError with no detail message.

public StackOverflowError(String s)

Constructs a StackOverflowError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.59 Class ThreadDeath

```
public class java.lang.ThreadDeath
    extends java.lang.Error (I-§1.48)
{
    // Constructors
    public ThreadDeath(); §1.59.1
}
```

An instance of ThreadDeath is thrown in the victim thread when the stop method with zero arguments (I-§1.19.37) in class Thread is called.

An application should catch instances of this class only if it must clean up after being terminated asynchronously. If ThreadDeath is caught by a method, it is important that it be rethrown so that the thread actually dies.

The top-level error handler does not print out a message if ThreadDeath is never caught.

The class ThreadDeath is specifically a subclass of Error rather than Exception, even though it is a "normally occurrence," because many applications catch all occurrences of Exception and then discard the exception.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ThreadDeath.ThreadDeath

public ThreadDeath ()

Constructs a new ThreadDeath object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.60 Class UnknownError

```
public class java.lang.UnknownError
    extends java.lang.VirtualMachineError (I-§1.63)
{
    // Constructors
    public UnknownError(); §1.60.1
    public UnknownError(String s); §1.60.2
}
```

Thrown when an unknown but serious exception has occurred in the Java Virtual Machine.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


UnknownError.UnknownError

public UnknownError()

Constructs an UnknownError with no detail message.

public UnknownError(String s)

Constructs an UnknownError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.61 Class UnsatisfiedLinkError

```
public class java.lang.UnsatisfiedLinkError
    extends java.lang.LinkageError (l-§1.53)
{
    // Constructors
    public UnsatisfiedLinkError(); §1.61.1
    public UnsatisfiedLinkError(String s); §1.61.2
}
```

Thrown if the The Java Virtual Machine cannot find an appropriate native-language definition of a method declared native.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

UnsatisfiedLinkError.UnsatisfiedLinkError

public UnsatisfiedLinkError()

Constructs an UnsatisfiedLinkError with no detail message.

public UnsatisfiedLinkError(String s)

Constructs an UnsatisfiedLinkError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.62 Class VerifyError

```
public class java.lang.VerifyError  
    extends java.lang.LinkageError (I-§1.53)  
{  
    // Constructors  
    public VerifyError(); §1.62.1  
    public VerifyError(String s); §1.62.2  
}
```

Thrown when the "verifier" detects that a class file, though well-formed, contains some sort of internal inconsistency or security problem.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

VerifyError.VerifyError

public VerifyError()

Constructs an VerifyError with no detail message.

public VerifyError(String s)

Constructs an VerifyError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.63 Class VirtualMachineError

```
public class java.lang.VirtualMachineError
    extends java.lang.Error (I-§1.48)
{
    // Constructors
    public VirtualMachineError(); §1.63.1
    public VirtualMachineError(String s); §1.63.2
}
```

Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


VirtualMachineError.VirtualMachineError

public VirtualMachineError()

Constructs a VirtualMachineError with no detail message.

public VirtualMachineError(String s)

Constructs a VirtualMachineError with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.io

Classes

- 2.1 Class BufferedInputStream
- 2.2 Class BufferedOutputStream
- 2.3 Class ByteArrayInputStream
- 2.4 Class ByteArrayOutputStream
- 2.5 Class DataInputStream
- 2.6 Class DataOutputStream
- 2.7 Class File
- 2.8 Class FileDescriptor
- 2.9 Class FileInputStream
- 2.10 Class FileOutputStream
- 2.11 Class FilterInputStream
- 2.12 Class FilterOutputStream
- 2.13 Class InputStream
- 2.14 ClassLineNumberInputStream
- 2.15 Class OutputStream
- 2.16 Class PipedInputStream
- 2.17 Class PipedOutputStream
- 2.18 Class PrintStream
- 2.19 Class PushbackInputStream
- 2.20 Class RandomAccessFile
- 2.21 Class SequenceInputStream
- 2.22 Class StreamTokenizer
- 2.23 Class StringBufferInputStream

Interfaces

- 2.24 Interface DataInput
- 2.25 Interface DataOutput
- 2.26 Interface FilenameFilter

Exceptions

- 2.27 Class EOFException
- 2.28 Class FileNotFoundException
- 2.29 Class IOException
- 2.30 Class InterruptedIOException
- 2.31 Class UTFDataFormatException


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.1 Class BufferedInputStream

```
public class java.io.BufferedInputStream
    extends java.io.FilterInputStream (l-§2.11)
{
    // Fields
    protected byte buf[];      §2.1.1
    protected int count;        §2.1.2
    protected int marklimit;    §2.1.3
    protected int markpos;      §2.1.4
    protected int pos;          §2.1.5

    // Constructors
    public BufferedInputStream(InputStream in);      §2.1.6
    public BufferedInputStream(InputStream in, int size); §2.1.7
    public int available();      §2.1.8

    // Methods
    public void mark(int readlimit);    §2.1.9
    public boolean markSupported();    §2.1.10
    public int read();                 §2.1.11
    public int read(byte b[], int off, int len); §2.1.12
    public void reset();               §2.1.13
    public long skip(long n);           §2.1.14
}
```

The class implements a buffered input stream. By setting up a such an input stream, an application can read bytes from a stream without necessarily causing a call to the underlying system for each byte read. The data is read by blocks into a buffer; subsequent reads can access the data directly from the buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.buf

protected byte buf[]

The buffer where data is stored.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.count

protected int count

The index one greater than the index of the last valid byte in the buffer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BufferedInputStream.marklimit

protected int marklimit

The maximum read ahead allowed after a call to the mark method (I-§2.1.9) before subsequent calls to the reset method (I-§2.1.13) fail.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.markpos

protected int markpos

The value of the pos field ([I-§2.1.5](#)) at the time the last mark [method](#) ([I-§2.1.9](#)) was called.
The value of this field is -1 if there is no current mark.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.pos

protected int pos

The current position in the buffer. This is the index of the next character to be read from the buffer ([I-§2.1.1](#)) array.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.BufferedInputStream

public BufferedInputStream(InputStream in)

Creates a new buffered input stream to read data from the specified input stream with a default 512-byte buffer size.

Parameters:

in- the underlying input stream

public BufferedInputStream(InputStream in, int size)

Creates a new buffered input stream to read data from the specified input stream with the specified buffer size.

Parameters:

in- the underlying input stream

size- the buffer size

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.available

public int available()
throws IOException

Determines the number of bytes that can be read from this input stream without blocking.

The available method of BufferedInputStream returns the sum of the the number of bytes remaining to be read in the buffer (`count - pos`) and the result of calling the available method of the underlying input stream (I-§2.11.1).

Returns:

the number of bytes that can be read from this input stream without blocking.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

available in class FilterInputStream (I-§2.11.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.mark

public void mark(int readlimit)

Marks the current position in this input stream. A subsequent call to the reset method (I-§2.1.13) repositions the stream at the last marked position so that subsequent reads re-read the same bytes.

The readlimit arguments tells the input stream to allow that many bytes to be read before the mark position gets invalidated.

Parameters:

readlimit- the maximum limit of bytes that can be read before the mark position becomes invalid.

Overrides:

mark in class FilterInputStream (I-§2.11.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.markSupported

public boolean markSupported()

Determines if this input stream supports the mark ([I-§2.13.4](#)) and reset ([I-§2.13.9](#)) methods. The markSupported method of Buffered Input stream returns true.

Returns:

a boolean indicating if this stream type supports the mark and reset methods.

Overrides:

markSupported in class FilterInputStream ([I-§2.11.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.read

public int read()
throws IOException

Reads the next byte of data from this buffered input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until either input data is available, the end of the stream is detected, or an exception is thrown.

The read method of BufferedInputStream returns the next byte of data from its buffer if the buffer isn't empty. Otherwise, it refills the buffer from the underlying input stream (I-§2.11.1) and returns the next character, if the underlying stream hasn't returned an end-of-stream indicator.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class FilterInputStream (I-§2.11.7).

public int read(byte b[], int off, int len)
throws IOException

Reads up to len bytes of data from this buffered input stream into an array of bytes. This method blocks until some input is available.

The read method of BufferedInputStream copies bytes from its buffer into the array argument if the buffer isn't empty. Otherwise, it refills the buffer from the underlying input stream (I-§2.11.1) and unless the underlying stream returns an end-of-stream indicator, it fills the array argument with characters from the newly filled buffer.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class FilterInputStream ([I-§2.11.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BufferedInputStream.reset

public void reset()
throws IOException

Repositions this stream to the position at the time the mark method (I-§2.1.9) was last called on this input stream.

Throws

IOException (I-§2.29)

If this stream has not been marked or if the mark has been invalidated.

Overrides:

reset in class FilterInputStream (I-§2.11.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedInputStream.skip

public long skip(long n)
throws IOException

Skips over and discards n bytes of data from the input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly zero. The actual number of bytes skipped is returned.

The skip method of BufferedInputStream compares the number of bytes it has available in its buffer, a, where $a = \text{count} - \text{pos}$, with n. If $a = \text{count} - \text{pos}$, then the pos field is incremented by n. Otherwise, the pos field is incremented to have the value count, and the remaining bytes (if any) are skipped by calling the underlying input stream's (I-§2.11.1) skip method with the argument $n - a$.

Parameters:

n- the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

skip in class FilterInputStream (I-§2.11.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.2 Class BufferedOutputStream

```
public class java.io.BufferedOutputStream
    extends java.io.FilterOutputStream (l-§2.12)
{
    // Fields
    protected byte buf[];      §2.2.1
    protected int count;      §2.2.2

    // Constructors
    public BufferedOutputStream(OutputStream out); §2.2.3
    public BufferedOutputStream(OutputStream out, int size); §2.2.4

    // Methods
    public void flush();      §2.2.5
    public void write(byte b[], int off, int len); §2.2.6
    public void write(int b); §2.2.7
}
```

The class implements a buffered output stream. By setting up a such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written. The data is written into a buffer, and then written to the underlying stream if the buffer reaches its capacity, if the buffer output stream is closed, or if the buffer output stream is explicitly flushed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedOutputStream.buf

protected byte buf[]

The buffer where data is stored.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedOutputStream.count

protected int count

The number of valid bytes in the buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedOutputStream.BufferedOutputStream

public BufferedOutputStream(OutputStream out)

Creates a new buffered output stream to write data to the specified underlying output stream with a default 512-byte buffer size.

Parameters:

out- the underlying output stream

public BufferedOutputStream(OutputStream out, int size)

Creates a new buffered output stream to write data to the specified underlying output stream with the specified buffer size.

Parameters:

out- the underlying output stream

size- the buffer size

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedOutputStream.flush

public void flush()
throws IOException

Flushes this buffered output stream. This forces any buffered output bytes to be written out to the underlying output stream [\(I-§2.12.1\)](#).

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

Overrides:

flush in class FilterOutputStream [\(I-§2.12.4\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BufferedOutputStream.write

public void write(byte b[], int off, int len)
throws IOException

Writes len bytes from the specified byte array starting at offset off to this buffered output stream.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class FilterOutputStream ([I-§2.12.6](#)).

public void write(int b)
throws IOException

Writes the specified byte to this buffered output stream.

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class FilterOutputStream ([I-§2.12.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.3 Class ByteArrayInputStream

```
public class java.io.ByteArrayInputStream
    extends java.io.InputStream (l-§2.13)
{
    // Fields
    protected byte buf[]; §2.3.1
    protected int count; §2.3.2
    protected int pos; §2.3.3

    // Constructors
    public ByteArrayInputStream(byte buf[]); §2.3.4
    public ByteArrayInputStream(byte buf[], int offset, int length);
                                                §2.3.5

    // Methods
    public int available(); §2.3.6
    public int read(); §2.3.7
    public int read(byte b[], int off, int len); §2.3.8
    public void reset(); §2.3.9
    public long skip(long n); §2.3.10
}
```

This class allows an application to create an input stream in which the bytes read are supplied by the contents of a byte array. Applications can also read bytes from a string by using a StringBufferInputStream (l-§2.23).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ByteArrayInputStream.buf

protected byte buf[]

The byte array containing the data.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.count

protected int count

The index one greater than the last valid character in the input stream buffer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ByteArrayInputStream.pos

protected int pos

The index of the next character to read from the input stream buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.ByteArrayInputStream

public **ByteArrayInputStream**(byte buf[])

Creates a new byte array input stream which reads data from the specified byte array. The byte array is not copied.

Parameters:

buf- the input buffer

public **ByteArrayInputStream**(byte buf[], int offset,int length)

Creates a new byte array input stream which reads data from the specified byte array. Up to length characters are to be read from the byte array, starting at the indicated offset.

The byte array is not copied.

Parameters:

buf- the input buffer

offset- the offset in the buffer of the first byte to read

length- the maximum number of bytes to read from the buffer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.available

public `int available()`

Determines the number of bytes that can be read from this input stream without blocking.

The available method of `ByteArrayInputStream` returns the value of `count - pos`, which is the number of bytes remaining to be read from the input buffer.

Returns:

the number of bytes that can be read from the input stream without blocking.

Overrides:

available in class `InputStream` ([I-§2.13.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.read

public **int** **read()**

Reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the the end of the stream has been reached, the value -1 is returned.

The read method of ByteArrayInputStream cannot block.

Returns:

the next byte of data, or -1 if the end of the stream has been reached.

Overrides:

read in class InputStream (I-§2.13.6).

public **int** **read(byte b[], int off, int len)**

Reads up to len bytes of data into an array of bytes from this input stream. This read method cannot block.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Overrides:

read in class InputStream (I-§2.13.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.reset

public void reset()

Resets this input stream to begin reading from the same position at which it first started reading from the buffer.

Overrides:

reset in class InputStream ([I-§2.13.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayInputStream.skip

public long skip(long n)

Skips n bytes of input from this input stream. Fewer bytes might be skipped if the end of the input stream is reached.

Parameters:

n- the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Overrides:

skip in class InputStream ([I-§2.13.10](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.4 Class ByteArrayOutputStream

```
public class java.io.ByteArrayOutputStream
    extends java.io.OutputStream (l-§2.15)
{
    // Fields
    protected byte buf[];      §2.4.1
    protected int count;      §2.4.2

    // Constructors
    public ByteArrayOutputStream(); §2.4.3
    public ByteArrayOutputStream(int size); §2.4.4

    // Methods
    public void reset();      §2.4.5
    public int size(); §2.4.6
    public byte[] toByteArray(); §2.4.7
    public String toString(); §2.4.8
    public String toString(int hiByte); §2.4.9
    public void write(byte b[], int off, int len); §2.4.10
    public void write(int b); §2.4.11
    public void writeTo(OutputStream out); §2.4.12
}
```

This class implements an output stream in which the data is written into a byte array. The buffer automatically grows as data is written to it.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ByteArrayOutputStream.buf

protected byte buf[]

The buffer where data is stored.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.count

protected int count

The number of valid bytes in the buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.ByteArrayOutputStream

public `ByteArrayOutputStream()`

Creates a new byte array output stream.

public `ByteArrayOutputStream(int size)`

Creates a new byte array output stream. The buffer capacity is initially 32 bytes, though its size increases if necessary.

Parameters:

`size`- the initial size

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.reset

public void reset()

Resets the count field ([I-§2.3.2](#)) of this byte array output stream to zero, so that all currently accumulated output in the output stream is discarded. The output stream can be used again, reusing the already allocated buffer space.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.size

public int size()

Returns:

the value of the count field ([I-§2.4.2](#)), which is the number of valid bytes in this output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.toByteArray

public byte[] toByteArray()

Creates a newly allocated byte array whose size is the current size of this output stream (I-§2.4.6) and into which the valid contents of the buffer have been copied.

Returns:

the current contents of this output stream, as a byte array.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.toString

public String toString()

Creates a newly allocated string whose size is the current size of this output stream (I-§2.4.6) and into which the valid contents of the buffer have been copied. Each character *c* in the resulting string is constructed from the corresponding element *b* in the byte array such that:

$c == (\text{char})(b \ \& \ 0\text{xff})$

Returns:

the current contents of this output stream, as a string.

Overrides:

toString in class Object (I-§1.12.9).

public String toString(int hbyte)

Creates a newly allocated string whose size is the current size of the output stream (I-§2.4.6) and into which the valid contents of the buffer have been copied. Each character *c* in the resulting string is constructed from the corresponding element *b* in the byte array such that:

$c == (\text{char})(((\text{hbyte} \ \& \ 0\text{xff}) \ll 8) \mid (b \ \& \ 0\text{xff}))$

Parameters:

hbyte- the bits set

Returns:

the current contents of the output stream, as a string.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.write

public void write(byte b[], int off, int len)

Writes len bytes from the specified byte array starting at offset off to this byte array output stream.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Overrides:

write in class OutputStream (I-§2.15.5).

public void write(int b)

Writes the specified byte to this byte array output stream.

Parameters:

b- the byte to be written

Overrides:

write in class OutputStream (I-§2.15.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ByteArrayOutputStream.writeTo

public void writeTo(OutputStream out)
throws IOException

Writes the complete contents of this byte array output stream to the specified output stream argument, as if by calling the output stream's write method using `out.write(buf, 0, count)`.

Parameters:

`out`– the output stream to which to write the data

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.5 Class DataInputStream

```
public class java.io.DataInputStream
    extends java.io.FilterInputStream (I-§2.11)
    implements java.io.DataInput (I-§2.24)
{
    // Constructors
    public DataInputStream(InputStream in); §2.5.1

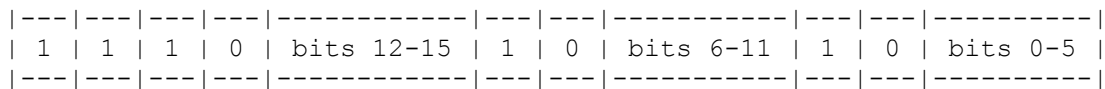
    // Methods
    public final int read(byte b[]); §2.5.2
    public final int read(byte b[], int off, int len); §2.5.3
    public final boolean readBoolean(); §2.5.4
    public final byte readByte(); §2.5.5
    public final char readChar(); §2.5.6
    public final double readDouble(); §2.5.7
    public final float readFloat(); §2.5.8
    public final void readFully(byte b[]); §2.5.9
    public final void readFully(byte b[], int off, int len); §2.5.10
    public final int readInt(); §2.5.11
    public final String readLine(); §2.5.12
    public final long readLong(); §2.5.13
    public final short readShort(); §2.5.14
    public final int readUnsignedByte(); §2.5.15
    public final int readUnsignedShort(); §2.5.16
    public final String readUTF(); §2.5.17
    public final static String readUTF(DataInput in); §2.5.18
    public final int skipBytes(int n); §2.5.19
}
```

A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream (I-§2.6) to write data which can later be read by a data input stream.

Data input streams and data output streams represent Unicode strings in a format that is a slight modification of UTF-8. 1All characters in the range 'u0001' to 'u007F' are represented by a single byte:

```
|---|-----|
| 0 | bits 0-7 |
|---|-----|
```

```
|---|---|---|-----|---|---|-----|
| 1 | 1 | 0 | bits 6-10 | 1 | 0 | bits 0-5 |
|---|---|---|-----|---|---|-----|
```

The two differences between this format and the "standard" UTF-8 format are the following:

- The null byte '\u0000' is encoded in two-byte format rather than one-byte, so that the encoded strings never have embedded nulls.
- Only the one-byte, two-byte, and three-byte formats are used.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInputStream.DataInputStream

public DataInputStream(InputStream in)

Creates a new data input stream to read data from the specified input stream.

Parameters:

`in` - the input stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.read

public **final** **int** **read**(**byte** **b[]**)
throws **IOException**

Reads up to `byte.length` bytes of data from this data input stream into an array of bytes. This method blocks until some input is available.

The read method of `DataInputStream` calls the read method (I-§2.13.8) of its underlying input stream (I-§2.11.1) with the three arguments `b`, `0`, and `b.length`, and returns whatever value that method returns.

Parameters:

`b`– the buffer into which the data is read

Returns:

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Throws

`IOException` (I-§2.29)

If an I/O error occurs.

Overrides:

read in class `FilterInputStream` (I-§2.11.8).

public **final** **int** **read**(**byte** **b[]**, **int** **off**, **int** **len**)
throws **IOException**

Reads up to `len` bytes of data from this data input stream into an array of bytes. This method blocks until some input is available.

The read method of `DataInputStream` calls the read method (I-§2.13.8) of its underlying input stream (I-§2.11.1) with the same arguments and returns whatever value that method returns.

Parameters:

`b`– the buffer into which the data is read

`off`– the start offset of the data

`len`– the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class FilterInputStream ([I-§2.11.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInputStream.readBoolean

public final boolean readBoolean()
throws IOException

Reads a boolean from this data input stream. This method reads a single byte from the underlying input stream (I-§2.11.1). A value of 0 represents false. Any other value represents true. This method blocks until either the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the boolean value read.

Throws

EOFException (I-§2.24)

If this input stream has reached the end.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readByte

public final byte readByte()
throws IOException

Reads a signed 8-bit value from this data input stream. This method reads a byte from the underlying input stream (I-§2.11.1). If the byte read is b, where {ewc msdncl, EWGraphic, IOS4bp 0 /a "sunref.BMP"}, then the result is

(byte) (b)

This method blocks until either the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of this input stream as a signed 8-bit byte.

Throws

EOFException (I-§2.24)

If this input stream has reached the end.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

DataInputStream.readChar

public final char readChar()
throws IOException

Reads a Unicode character from this data input stream. This method reads two bytes from the underlying input stream (I-§2.11.1). If the bytes read, in order, are b1 and b2, where {ewc msdn cd, EWGraphic, IOS5bp 0 /a "sunref.BMP"}, then the result is equal to

`(char) ((b1 << 8) | b2)`

This method **blocks** until either the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this input stream as an Unicode character.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdn cd.dll, ewcright, /c"Microsoft"}

DataInputStream.readDouble

public **final** **double** **readDouble()**
throws **IOException**

Reads a double from this data input stream. This method reads a long value as if by the readLong method (I-§2.5.13) and then converts that long to a double using the long-Bits-To-Double method (I-§1.6.18) in class Double.

This method blocks until either the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next eight bytes of this input stream, interpreted as a double.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading eight bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readFloat

public **final** **float** **readFloat()**
throws **IOException**

Reads a float from this data input stream. This method reads an int value as if by the readInt method (I-§2.5.11) and then converts that int to a float using the intBitsToFloat method (I-§1.7.14) in class Float. This method blocks until either the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next four bytes of this input stream, interpreted as a float.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading four bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readFully

public final void readFully(byte b[])
throws IOException

Reads b.length bytes from this data input stream into the byte array. This method reads repeatedly from the underlying stream (I-§2.11.1) until all the bytes are read. This method blocks until either all the bytes are read, the end of the stream is detected, or an exception is thrown.

Parameters:

b- the buffer into which the data is read

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading all the bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

public final void readFully(byte b[], int off, int len)
throws IOException

Reads exactly len bytes from this data input stream into the byte array. This method reads repeatedly from the underlying stream (I-§2.11.1) until all the bytes are read. This method blocks until either all the bytes are read, the end of the stream is detected, or an exception is thrown.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the number of bytes to read read

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading all the bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInputStream.readInt

public final int readInt()
throws IOException

Reads a signed 32-bit integer from this data input stream. This method reads four bytes from the underlying input stream (I-§2.11.1). If the bytes read, in order, are b1, b2, b3, and b4 where {ewc msdn cd, EWGraphic, IOS9bp 0 /a "sunref.BMP"}, then the result is equal to

$(b1 \ll 24) \mid (b2 \ll 16) + (b3 \ll 8) + b4$

This method blocks until either the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next four bytes of this input stream, interpreted as an int.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdn cd, ewcright, /c"Microsoft"}

DataInputStream.readLine

public **final** **String** **readLine()**
throws **IOException**

Reads the next line of text from this data input stream. This method successively reads bytes from the underlying input stream (I-§2.11.1) until it reaches the end of a line of text.

A line of text is terminated by a carriage return character ('\r'), a newline character ('\n'), a carriage return character immediately followed by a newline character, or the end of the input stream. The line-terminating character(s), if any, are included as part of the string returned.

This method blocks until either a newline character is read, a carriage return and the byte following it are read (to see if it is a newline), the end of the stream is detected, or an exception is thrown.

Returns:

the next line of text from this input stream.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readLong

public final long readLong()
throws IOException

Reads a signed 64-bit integer from this data input stream. This method reads eight bytes from the underlying input stream (I-§2.11.1). If the bytes read, in order, are b1, b2, b3, b4, b5, b6, b7, and b8, where

{ewc msdncl, EWGraphic, IOS11bp 0 /a "sunref.BMP"},

then the result is equal to

$((\text{long})b1 \ll 56) + ((\text{long})b2 \ll 48) + ((\text{long})b3 \ll 40) + ((\text{long})b4 \ll 32) + ((\text{long})b5 \ll 24) + ((\text{long})b6 \ll 16) + ((\text{long})b7 \ll 8) + (\text{long})b8$

This method blocks until either the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next eight bytes of this input stream, interpreted as a long.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading eight bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

DataInputStream.readShort

public final short readShort()
throws IOException

Reads a signed 16-bit number from this data input stream. The method reads two bytes from the underlying input stream (I-§2.11.1). If the two bytes read, in order, are b1 and b2, where each of the two values is between 0 and 255, inclusive, then the result is equal to:

$$(\text{short})((b1 \ll 8) | b2)$$

This method blocks until either the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this input stream, interpreted as a signed 16-bit number.

Throws

EOFException (I-§2.24)

If this input stream reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readUnsignedByte

public **final** **int** **readUnsignedByte()**
throws **IOException**

Reads an unsigned 8-bit number from this data input stream. This method reads a byte from this data input stream's underlying input stream (I-§2.11.1) and returns that byte. This method blocks until either the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of this input stream, interpreted as an unsigned 8-bit number.

Throws

EOFException (I-§2.24)

If this input stream has reached the end.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.readUnsignedShort

public **final** **int** **readUnsignedShort()**
throws **IOException**

Reads an unsigned 16-bit number from this data input stream. This method reads two bytes from the underlying input stream ([I-§2.11.1](#)). If the bytes read, in order, are b1 and b2, where {ewc msdncl, EWGraphic, IOS14bp 0 /a "sunref.BMP"}, then the result is equal to

$(b1 \ll 8) \mid b2$

This method blocks until either the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this input stream, interpreted as an unsigned 16-bit integer.

Throws

EOFException ([I-§2.24](#))

If this input stream reaches the end before reading two bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

DataInputStream.readUTF

public final String readUTF()
throws IOException

Reads in a string which has been encoded using a modified UTF-8 format from this data input stream. This method calls readUTF(this). See the following method for a more complete description of the format.

This method blocks until either all the bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

a Unicode string.

Throws

EOFException ([I-§2.24](#))

If this input stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public final static String readUTF(DataInput in)
throws IOException

Reads in a string from the specified data input stream. The string has been encoded using a modified using a modified UTF-8 format.

The first two bytes from are read as if by readUnsignedShort ([I-§2.5.16](#)). This values gives the number of following bytes that are in the encoded string. (Note: *not* the length of the resuling string). The following bytes are then interpreted as bytes encoding characters in the UTF-8 format ([page I-225](#)) and are converted into characters.

This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

Parameters:

`in`- a data input stream

Returns:

a Unicode string.

Throws

EOFException ([I-§2.24](#))

If the input stream reaches the end before all the bytes.

Throws

UTFDataFormatException ([I-§2.31](#))

If the bytes do not represent a valid UTF-8 encoding of a Unicode string.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInputStream.skipBytes

public final int skipBytes(int n)
throws IOException

Skips exactly n bytes of input in the underlying input stream ([I-§2.11.1](#)). This method blocks until either all the bytes are skipped, the end of the stream is detected, or an exception is thrown.

Parameters:

n- the number of bytes to be skipped

Returns:

the number of bytes skipped, which is always n.

Throws

EOFException ([I-§2.24](#))

If this input stream reaches the end before skipping all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹X/Open Company Ltd., "File System Safe UCS Transformation Format (FSS_UTF)", X/Open Preliminary Specification, Document Number: P316. This information also appears in ISO/IEC 10646, Annex P.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.6 Class DataOutputStream

```
public class java.io.DataOutputStream
    extends java.io.FilterOutputStream (I-§2.12)
    implements java.io.DataOutput (I-§2.25)
{
    // Fields
    protected int written; §2.6.1

    // Constructors
    public DataOutputStream(OutputStream out); §2.6.2

    // Methods
    public void flush(); §2.6.3
    public final int size(); §2.6.4
    public void write(byte b[], int off, int len); §2.6.5
    public void write(int b); §2.6.6
    public final void writeBoolean(boolean v); §2.6.7
    public final void writeByte(int v); §2.6.8
    public final void writeBytes(String s); §2.6.9
    public final void writeChar(int v); §2.6.10
    public final void writeChars(String s); §2.6.11
    public final void writeDouble(double v); §2.6.12
    public final void writeFloat(float v); §2.6.13
    public final void writeInt(int v); §2.6.14
    public final void writeLong(long v); §2.6.15
    public final void writeShort(int v); §2.6.16
    public final void writeUTF(String str); §2.6.17
}
```

A data input stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream (I-§2.5) to read the data back in.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.written

protected int written

The number of bytes written to the data output stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.DataOutputStream

public `DataOutputStream`(`OutputStream` out)

Creates a new data output stream to write data to the specified underlying output stream ([1-§2.12.1](#)).

Parameters:

out – the underlying output stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.flush

public void flush()
throws IOException

Flushes this data output stream. This forces any buffered output bytes to be written out to the stream.

The flush method of DataOutputStream calls the flush method ([I-§2.15.3](#)) of its underlying output stream ([I-§2.12.1](#)).

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

flush in class FilterOutputStream ([I-§2.12.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.size

public final int size()

Determines the number of bytes written to this data output stream.

Returns:

The value of the written field (I-§2.6.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.write

```
public void write(byte b[], int off, int len)
throws IOException
```

Writes len bytes from the specified byte array starting at offset off to the underlying output stream ([I-§2.12.1](#)).

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class FilterOutputStream ([I-§2.12.6](#)).

```
public void write(int b)
throws IOException
```

Writes the specified byte to the underlying output stream ([I-§2.12.1](#)).

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class FilterOutputStream ([I-§2.12.7](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeBoolean

```
public final void writeBoolean(boolean v)  
throws IOException
```

Writes a boolean to the underlying output stream ([I-§2.12.1](#)) as a one-byte value. The value true is written out as the value (byte) 1; the value false is written out as the value (byte) 0.

Parameters:

v— a boolean value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.writeByte

```
public final void writeByte(int v)  
throws IOException
```

Writes out a byte to the underlying output stream ([I-§2.12.1](#)) as a one-byte value.

Parameters:

`v` – a byte value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeBytes

public final void writeBytes(String s)
throws IOException

Writes out the string to the underlying output stream ([I-§2.12.1](#)) as a sequence of bytes. Each character in the string is written out, in sequence, by discarding its high eight bits.

Parameters:

s – a string of bytes to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.writeChar

```
public final void writeChar(int v)  
throws IOException
```

Writes a char to the underlying output stream ([I-§2.12.1](#)) as a two-byte value, high byte first.

Parameters:

`v` – a char value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeChars

public **final** **void** **writeChars**(String s)
throws IOException

Writes a string to the underlying output stream ([I-§2.12.1](#)) as a sequence of characters. Each character is written to the data output stream as if by the writeChar [method](#) ([I-§2.6.10](#)).

Parameters:

s- a String value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.writeDouble

```
public final void writeDouble(double v)  
throws IOException
```

Converts the double argument to a long using the doubleToLongBits method (I-§1.6.8) in class Double, and then writes that long value to the underlying output stream (I-§2.12.1) as an eight-byte quantity, high-byte first.

Parameters:

v – a double value to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.writeFloat

```
public final void writeFloat(float v)  
throws IOException
```

Converts the float argument to an int using the floatToIntBits method (I-§1.7.11) in class Float, and then writes that int value to the underlying output stream (I-§2.12.1) as a four-byte quantity, high-byte first.

Parameters:

v – a float value to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutputStream.writeInt

```
public final void writeInt(int v)  
throws IOException
```

Writes an int to the underlying output stream ([I-§2.12.1](#)) as four bytes, high-byte first.

Parameters:

v – an int to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeLong

```
public final void writeLong(long v)  
throws IOException
```

Writes a long to the underlying output stream ([I-§2.12.1](#)) as eight bytes, high-byte first.

Parameters:

v – a long to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeShort

```
public final void writeShort(int v)  
throws IOException
```

Writes a short to the underlying output stream ([I-§2.12.1](#)) as two bytes, high-byte first.

Parameters:

`v` – a short to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutputStream.writeUTF

public final void writeUTF(String str)
throws IOException

Writes out a string to the underlying output stream ([I-§2.12.1](#)) using UTF-8 encoding in a machine-independent manner.

First, two bytes are written to the output stream as if by the writeShort method ([I-§2.6.16](#)) giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the UTF-8 encoding for each character.

Parameters:

`str`— a string to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.7 Class File

```
public class java.io.File
    extends java.lang.Object (I-§1.12)
{
    // Fields
    public final static String pathSeparator; §2.7.1
    public final static char pathSeparatorChar; §2.7.2
    public final static String separator; §2.7.3
    public final static char separatorChar; §2.7.4

    // Constructors
    public File(File dir, String name); §2.7.5
    public File(String path); §2.7.6
    public File(String path, String name); §2.7.7

    // Methods
    public boolean canRead(); §2.7.8
    public boolean canWrite(); §2.7.9
    public boolean delete(); §2.7.10
    public boolean equals(Object obj); §2.7.11
    public boolean exists(); §2.7.12
    public String getAbsolutePath(); §2.7.13
    public String getName(); §2.7.14
    public String getParent(); §2.7.15
    public String getPath(); §2.7.16
    public int hashCode(); §2.7.17
    public boolean isAbsolute(); §2.7.18
    public boolean isDirectory(); §2.7.19
    public boolean isFile(); §2.7.20
    public long lastModified(); §2.7.21
    public long length(); §2.7.22
    public String[] list(); §2.7.23
    public String[] list(FilenameFilter filter); §2.7.24
    public boolean mkdir(); §2.7.25
    public boolean mkdirs(); §2.7.26
    public boolean renameTo(File dest); §2.7.27
    public String toString(); §2.7.28
}
```

Instances of this class represent the name of a file or directory on the host file system. A file is specified by a path name, which can either be an absolute path name or a path name relative to the current working directory. The path name must follow the naming conventions of the host platform.

The File class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File.pathSeparator

```
public final static String pathSeparator
```

The system-dependent path separator string. This field is initialized to contain the value of the system property (I-§1.18.9) "path.-separator."

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File.pathSeparatorChar

`public final static char pathSeparatorChar`

The system-dependent path separator character. This field is initialized to contain the first character of the value of the system property (I-§1.18.9) "path.separator". This character is often used to separate file names in a sequence of files given as a "path list."

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.separator

`public final static String separator`

The system-dependent path separator string. This field is initialized to contain the value of the system property (I-§1.18.9) "file.separator."

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.separatorChar

```
public final static char separatorChar
```

The system-dependent path separator string. This field is initialized to contain the first character of the value of the system property (I-§1.18.9)"file.separator." This character separates the directory and file components in a file name.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File.File

public File(File dir, String name)

Creates a File instance that represents the file with the specified name in the specified directory.

If the directory argument is null, the resulting File instance represents a file in the (system-dependent) current directory whose path name (I-§2.7.16) is the name argument. Otherwise, the File instance represents a file whose path name is the pathname of the directory, followed by the separator character (I-§2.7.3), followed by the name argument.

Parameters:

`dir`- the directory

`name`- the file path name

public File(String path)

Creates a File instance that represents the file whose path name (I-§2.7.16) is the given path argument.

Parameters:

`path`- the file path name

Throws

NullPointerException (I-§1.40)

If the file path is equal to null.

public File(String path, String name)

Creates a File instance whose path name (I-§2.7.16) is the pathname of the specified directory, followed by the separator character (I-§2.7.3), followed by the name argument.

Parameters:

`path`- the directory path name

name- the file path name

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File.canRead

public boolean canRead()

Determines if the application can read from the specified file.

Returns:

true if the file specified by this object exists and the application can read the file; false otherwise.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkRead method ([I-§1.15.19](#)) is called with the path name ([I-§2.7.16](#)) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.canWrite

public boolean canWrite()

Determines if the application can write to this file.

Returns:

true if the application is allowed to write to a file whose name is specified by this object; false otherwise.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkWrite method ([I-§1.15.24](#)) is called with the path name ([I-§2.7.16](#)) of this file to see if the application is allowed write access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.delete

public boolean delete()

Deletes the file specified by this object.

Returns:

true if the file is successfully deleted; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkDelete method (I-§1.15.9) is called with with the path name (I-§2.7.16) of this file to see if the application is allowed to delete the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a File object whose path name is equal to the path name of this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.exists

public boolean exists()

Determines if this File exists.

Returns:

true if the file specified by this object exists; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.getAbsolutePath

public String getAbsolutePath()

If this object represents an absolute path name (I-§2.7.18), then returns the path name (I-§2.7.16). Otherwise, return a path name that is a concatenation of the current user directory, the separator character, and the path name of this file object.

The system property (I-§1.18.9) user.dir contains the current user directory.

Returns:

a system-dependent absolute path name for this file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.getName

public String getName()

Returns the name of the file represented by this object. The name is everything in the path name (I-§2.7.16) after the last occurrence of the separator character (I-§2.7.3).

Returns:

the name of the file (without any directory components) represented by this File object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.getParent

public String getParent()

Returns the parent directory of the file represented by this object. The parent directory is everything in the path name (I-§2.7.16) before the last occurrence of the separator character (I-§2.7.3), or null if the separator character does not appear in the path name.

Returns:

the name of the directory of the file represented by this File object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.getPath

public String getPath()

Returns:

the path name represented by this File object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


File.hashCode

public int hashCode()

Returns:

a hash code value for this File object.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.isAbsolute

public boolean isAbsolute()

Determines if this file represents an absolute path name. The definition of an absolute path name is system-dependent.1

Returns:

true if the path name (I-§2.7.16) indicated by the File object is an absolute path name;
false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.isDirectory

public boolean isDirectory()

Determines if the file represented by this File object is a directory.

Returns:

true if this file exists and is a directory; false otherwise.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkRead method ([I-§1.15.19](#)) is called with the path name ([I-§2.7.16](#)) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.isFile

public boolean isFile()

Determines if the file represented by this File object is a "normal" file.

A file is "normal" if it is not a directory and, in addition, satisfies other system-dependent criteria. Any non-directory file created by a Java application is guaranteed to be a normal file.

Returns:

true if the file specified by this object exists and is a "normal" file; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.lastModified

public long lastModified()

Determines the time that the file represented by this File object was last modified.

The return value is system-dependent and should only be used to compare with other values returned by last modified. It should not be interpreted as an absolute time.

Returns:

the time the file specified by this object was last modified, or 0L if the specified file does not exist.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.length

public long length()

Determines the length of the file represented by this File object.

Returns:

the length, in bytes, of the file specified by this object, or 0L if the specified file does not exist.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.list

public String[] list()

Lists the files in the directory specified by this File.

Returns:

an array of file names in the specified directory. This list does not include the current directory or the parent directory ("." and ".." on Unix systems).

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

public String[] list(FilenameFilter filter)

Lists the files in the directory specified by this file that satisfy the specified filter (I-§2.26).

Parameters:

filter- a filename filter

Returns:

an array of file names in the specified directory that satisfy the filter. This list does not include the current directory or the parent directory ("." and ".." on Unix systems).

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkRead method (I-§1.15.19) is called with the path name (I-§2.7.16) of this file to see if the application is allowed read access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.mkdir

public boolean mkdir()

Creates a directory whose path name is specified by this File object.

Returns:

true if the directory could be created; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkWrite method (I-§1.15.24) is called with the path name (I-§2.7.16) of this file to see if the application is allowed write access to the file.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.mkdirs

public boolean mkdirs()

Creates a directory whose path name is specified by this File object. In addition, creates all parent directories as necessary.

Returns:

true if the directory (or directories) could be created; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkWrite method (I-§1.15.24) is called with the path name (I-§2.7.16) of each of the directories that is to be created, before any of the directories are created.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.renameTo

public boolean renameTo(**File** dest)

Renames the file specified by this File object to have the path name given by the File argument.

Parameters:

dest – the new file name

Returns:

true if the renaming succeeds; false otherwise.

Throws

SecurityException (I-§1.43)

If a security manager exists, its checkWrite method (I-§1.15.24) is called both with the path name (I-§2.7.16) of this file object and with the path name of the destination target object to see if the application is allowed to write to both files.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

File.toString

public String toString()

Returns a string representation of this object.

Returns:

a string giving the path name (I-§2.7.16) of this object.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹For example, on Unix, a path name is absolute if its first character is the separator character (I-§2.7.3). On Windows platforms, an path name is absolute if its first character is an ASCII '\', '/', or if it begins with a letter followed by a colon.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.8 Class FileDescriptor

```
public final class java.io.FileDescriptor
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static FileDescriptor err;      §2.8.1
    public final static FileDescriptor in;      §2.8.2
    public final static FileDescriptor out;     §2.8.3

    // Constructors
    public FileDescriptor(); §2.8.4

    // Methods
    public boolean valid(); §2.8.5
}
```

Instances of the file descriptor class serve as an opaque handle to the underlying machine-specific structure representing an open file or an open socket.

Applications should not create their own file descriptors.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDescriptor.err

```
public final static FileDescriptor err
```

A handle to the standard error stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDescriptor.in

```
public final static FileDescriptor in
```

A handle to the standard input stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDescriptor.out

```
public final static FileDescriptor out
```

A handle to the standard output stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDescriptor.FileDescriptor

public FileDescriptor()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDescriptor.valid

public boolean valid()

Returns:

true if the file descriptor object represents a valid, open file or socket; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.9 Class FileInputStream

```
public class java.io.FileInputStream
    extends java.io.InputStream (l-§2.13)
{
    // Constructors
    public FileInputStream(File file); §2.9.1
    public FileInputStream(FileDescriptor fdObj); §2.9.2
    public FileInputStream(String name); §2.9.3

    // Methods
    public int available(); §2.9.4
    public void close(); §2.9.5
    protected void finalize(); §2.9.6
    public final FileDescriptor getFD(); §2.9.7
    public int read(); §2.9.8
    public int read(byte b[]); §2.9.9
    public int read(byte b[], int off, int len); §2.9.10
    public long skip(long n); §2.9.11
}
```

A file input stream is an input stream for reading data from a File (l-§2.7) or from a FileDescriptor (l-§2.8).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileInputStream.FileInputStream

public **FileInputStream**(**File** file)
throws **FileNotFoundException**

Creates an input file stream to read from the specified File object.

Parameters:

file- the file to be opened for reading

Throws

FileNotFoundException ([I-§2.28](#))

If the file is not found.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkRead method ([I-§1.15.19](#)) is called with the path name ([I-§2.7.16](#)) of this File argument to see if the application is allowed read access to the file. This may result in a security exception ([I-§1.43](#)).

public **FileInputStream**(**FileDescriptor** fdObj)

Creates an input file stream to read from the specified file descriptor.

Parameters:

fdObj- the file descriptor to be opened for reading

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkRead method ([I-§1.15.18](#)) is called with the file descriptor to see if the application is allowed to read from the specified file descriptor. This may result in a security exception ([I-§1.43](#)).

public **FileInputStream**(**String** name)
throws **FileNotFoundException**

Creates an input file stream to read from a file with the specified name.

If a security manager exists, its `checkRead` [method \(I-§1.15.19\)](#) is called with the name [argument](#) `t` to see if the application is allowed read access to the file. This may result in a security [exception \(I-§1.43\)](#).

Parameters:

`name`— the system dependent file name

Throws

`FileNotFoundException` [\(I-§2.28\)](#)

If the file is not found.

Throws

`SecurityException` [\(I-§1.43\)](#)

If a security manager exists, its `checkRead` [method \(I-§1.15.19\)](#) is called with the name [argument](#) `t` to see if the application is allowed read access to the file. This may result in a security [exception \(I-§1.43\)](#).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

FileInputStream.available

public int available()
throws IOException

Returns:

the number of bytes that can be read from this file input stream without blocking.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

Overrides:

available in class InputStream [\(I-§2.13.2\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileInputStream.close

public void close()
throws IOException

Closes this file input stream and releases any system resources associated with the stream.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

close in class InputStream ([I-§2.13.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileInputStream.finalize

protected void finalize()
throws IOException

This finalize method ensures that the close method (I-§2.9.5) of this file input stream is called when there are no more references to it.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

finalize in class Object (I-§1.12.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileInputStream.getFD

```
public final FileDescriptor getFD()  
throws IOException
```

Returns:

the file descriptor object (I-§2.8) associated with this stream.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileInputStream.read

public int read()
throws IOException

Reads a byte of data from this input stream. This method blocks if no input is available.

Returns:

the next byte of data, or -1 if the end of the file is reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.6](#)).

public int read(byte b[])
throws IOException

Reads up to b.length bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

Parameters:

b- the buffer into which the data is read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the file has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.7](#)).

public int read(byte b[], int off, int len)
throws IOException

Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

Parameters:

- `b`– the buffer into which the data is read
- `off`– the start offset of the data
- `len`– the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the file has been reached.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class InputStream (I-§2.13.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileInputStream.skip

```
public long skip(long n)  
throws IOException
```

Skips over and discards n bytes of data from the input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly zero. The actual number of bytes skipped is returned.

Parameters:

n– the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

skip in class InputStream ([I-§2.13.10](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.10 Class FileOutputStream

```
public class java.io.FileOutputStream
    extends java.io.OutputStream (l-§2.15)
{
    // Constructors
    public FileOutputStream(File file); §2.10.1
    public FileOutputStream(FileDescriptor fdObj); §2.10.2
    public FileOutputStream(String name); §2.10.3

    // Methods
    public void close(); §2.10.4
    protected void finalize(); §2.10.5
    public final FileDescriptor getFD(); §2.10.6
    public void write(byte b[]); §2.10.7
    public void write(byte b[], int off, int len); §2.10.8
    public void write(int b); §2.10.9
}
```

A file output stream is an output stream for writing data to a File (l-§2.7) or to a FileDescriptor (l-§2.8).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileOutputStream.FileOutputStream

public **FileOutputStream**(**File** file)
throws **IOException**

Creates an file output stream to write to the specified File object.

Parameters:

file- the file to be opened for writing

Throws

IOException ([I-§2.29](#))

If the file could not be opened for writing.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its `checkWrite` [method \(I-§1.15.24\)](#) is called with the path name ([I-§2.7.16](#)) of the File argument to see if the application is allowed write access to the file. This may result in a security exception (I-§1.43).

public **FileOutputStream**(**FileDescriptor** fdObj)

Creates an output file stream to write to the specified file descriptor.

Parameters:

fdObj- the file descriptor to be opened for writing

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its `checkWrite` [method \(I-§1.15.23\)](#) is called with the file descriptor to see if the application is allowed to write to the specified file descriptor. This may result in a security exception (I-§1.43).

public **FileOutputStream**(**String** name)
throws **IOException**

Creates an output file stream to write to the file with the specified name.

Parameters:

`name`– the system dependent file name

Throws

IOException ([I-§2.29](#))

If the file could not be opened for writing.

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its `checkWrite` [method](#) ([I-§1.15.24](#)) is called with the name [argument](#) to see if the application is allowed write access to the file. This may result in a security [exception](#) ([I-§1.43](#)).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

FileOutputStream.close

public void close()
throws IOException

Closes this file output stream and releases any system resources associated with this stream.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

Overrides:

close in class OutputStream [\(I-§2.15.2\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileOutputStream.finalize

protected void finalize()
throws IOException

This finalize method ensures that the close method (I-§2.9.5) of this file output stream is called when there are no more references to this stream.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

finalize in class Object (I-§1.12.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileOutputStream.getFD

public final FileDescriptor getFD()
throws IOException

Returns:

the file descriptor object (I-§2.8) associated with this stream.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileOutputStream.write

public void write(byte b[])
throws IOException

Writes b.length bytes from the specified byte array to this file output stream.

Parameters:

b- the data

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.4](#)).

public void write(byte b[], int off, int len)
throws IOException

Writes len bytes from the specified byte array starting at offset off to this file output stream.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.5](#)).

public void write(int b)
throws IOException

Writes the specified byte to this file output stream.

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.11 Class FilterInputStream

```
public class java.io.FilterInputStream
    extends java.io.InputStream (l-§2.13)
{
    // Fields
    protected InputStream in; §2.11.1

    // Constructors
    protected FilterInputStream(InputStream in); §2.11.2

    // Methods
    public int available(); §2.11.3
    public void close(); §2.11.4
    public void mark(int readlimit); §2.11.5
    public boolean markSupported(); §2.11.6
    public int read(); §2.11.7
    public int read(byte b[]); §2.11.8
    public int read(byte b[], int off, int len); §2.11.9
    public void reset(); §2.11.10
    public long skip(long n); §2.11.11
}
```

This class is the superclass of all classes that filter input streams. These streams sit on top of an already existing input stream (the *underlying* input stream), but provide additional functionality.

The class `FilterInputStream` itself simply overrides all methods of `InputStream` with versions that pass all requests to the underlying input stream. Subclasses of `FilterInputStream` may further override some of these methods as well as provide additional methods and fields.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FilterInputStream.in

protected InputStream in

The underlying input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.FilterInputStream

protected **FilterInputStream**(**InputStream** in)

Creates an input stream filter built on top of the specified input stream.

Parameters:

in- the underlying input stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.available

public int available()
throws IOException

Determines the number of bytes that can be read from this input stream without blocking.

The available method of FilterInputStream calls the available method of its underlying input stream (I-§2.11.1) and returns whatever value that method returns.

Returns:

the number of bytes that can be read from the input stream without blocking.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

available in class InputStream (I-§2.13.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.close

public void close()
throws IOException

Closes this input stream and releases any system resources associated with the stream. The close method of FilterInputStream calls the close method of its underlying input stream (I-§2.11.1).

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

close in class InputStream (I-§2.13.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.mark

public void mark(int readlimit)

Marks the current position in this input stream. A subsequent call to the reset method (I-§2.11.10) repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The readlimit arguments tells this input stream to allow that many bytes to be read before the mark position becomes invalid.

The mark method of FilterInputStream calls the mark method of its underlying input stream (I-§2.11.1) with the readlimit argument.

Parameters:

readlimit- the maximum limit of bytes that can be read before the mark position becomes invalid.

Overrides:

mark in class InputStream (I-§2.13.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.markSupported

public boolean markSupported()

Determines if this input stream supports the mark ([I-§2.13.4](#)) and reset ([I-§2.13.9](#)) methods. The markSupported method of FilterInputStream calls the markSupported method of its underlying input stream ([I-§2.11.1](#)) and returns whatever value that method returns.

Returns:

true if this stream type supports the mark and reset method; false otherwise.

Overrides:

markSupported in class InputStream ([I-§2.13.5](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.read

public int read()
throws IOException

Reads the next byte of data from this stream from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until either input data is available, the end of the stream is detected, or an exception is thrown.

The read method of FilterInputStream calls the read method of its underlying input stream ([I-§2.11.1](#)) and returns whatever value that method returns.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.6](#)).

public int read(byte b[])
throws IOException

Reads up to byte.length bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

The read method of FilterInputStream calls the read method of three arguments ([I-§2.11.9](#)) with the arguments b, 0, and b.length, and returns whatever value that method returns.

Note that this method does not call the one-argument read method of its underlying stream with the single argument b. Subclasses of FilterInputStream do not need to override this method if they have overridden the three-argument read method.

Parameters:

b- the buffer into which the data is read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.7](#)).

```
public int read(byte b[], int off, int len)
throws IOException
```

Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

The read method of FilterInputStream calls the read method of its underlying input stream ([I-§2.11.1](#)) with the same arguments and returns whatever value that method returns.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.8](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FilterInputStream.reset

public void reset()
throws IOException

Repositions this stream to the position at the time the mark method (I-§2.11.5) was last called on this input stream

The reset method of FilterInputStream calls the reset method of its underlying input stream (I-§2.11.1).

Throws

IOException (I-§2.29)

If the stream has not been marked or if the mark has been invalidated.

Overrides:

reset in class InputStream (I-§2.13.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterInputStream.skip

public long skip(long n)
throws IOException

Skips over and discards n bytes of data from the input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly zero. The actual number of bytes skipped is returned.

The skip method of FilterInputStream calls the skip method of its underlying input stream (I-§2.11.1) with the same argument, and returns whatever value that method does.

Parameters:

n– the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

skip in class InputStream (I-§2.13.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.12 Class FilterOutputStream

```
public class java.io.FilterOutputStream
    extends java.io.OutputStream (l-§2.15)
{
    // Fields
    protected OutputStream out; §2.12.1

    // Constructors
    public FilterOutputStream(OutputStream out); §2.12.2

    // Methods
    public void close(); §2.12.3
    public void flush(); §2.12.4
    public void write(byte b[]); §2.12.5
    public void write(byte b[], int off, int len); §2.12.6
    public void write(int b); §2.12.7
}
```

This class is the superclass of all classes that filter output streams. These streams sit on top of an already existing output stream (the *underlying* output stream), but provide additional functionality.

The class FilterOutputStream itself simply overrides all methods of OutputStream with versions that pass all requests to the underlying output stream. Subclasses of FilterOutputStream may further override some of these methods as well as provide additional methods and fields.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FilterOutputStream.out

protected OutputStream out

The underlying output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterOutputStream.FilterOutputStream

public FilterOutputStream(OutputStream out)

Creates an output stream filter built on top of the specified underlying output stream.

Parameters:

out – the underlying output stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterOutputStream.close

public void close()
throws IOException

Closes this output stream and releases any system resources associated with the stream.

The close method of FilterOutputStream calls its flush method ([I-§2.12.4](#)), and then calls the close method of its underlying output stream ([I-§2.12.1](#)).

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

close in class OutputStream ([I-§2.15.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterOutputStream.flush

public void flush()
throws IOException

Flushes this output stream and forces any buffered output bytes to be written out to the stream.

The flush method of FilterOutputStream calls the flush method of its underlying output stream (I-§2.12.1).

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

flush in class OutputStream (I-§2.15.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilterOutputStream.write

public void write(byte b[])
throws IOException

Writes b.length bytes to this output stream.

The write method of FilterOutputStream calls its write method of three arguments (I-§2.12.6) with the arguments b, 0, and b.length.

Note that this method does not call the one-argument write method of its underlying stream with the single argument b.

Parameters:

b- the data to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

write in class OutputStream (I-§2.15.4).

public void write(byte b[], int off, int len)
throws IOException

Writes len bytes from the specified byte array starting at offset off to this output stream.

The write method of FilterOutputStream calls the write method of one argument (I-§2.12.7) on each byte to output.

Note that this method does not call the write method of its underlying input stream with the same arguments. Subclasses of FilterOutputStream should provide a more efficient implementation of this method.

Parameters:

b- the data

`off`– the start offset in the data

`len`– the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.5](#)).

public void write(int b)
throws IOException

Writes the specified byte to this output stream.

The write method of FilterOutputStream calls the write method of its underlying output stream ([I-§2.12.1](#)).

Parameters:

`b`– the byte

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.13 Class InputStream

```
public abstract class java.io.InputStream
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public InputStream(); §2.13.1

    // Methods
    public int available(); §2.13.2
    public void close(); §2.13.3
    public void mark(int readlimit); §2.13.4
    public boolean markSupported(); §2.13.5
    public abstract int read(); §2.13.6
    public int read(byte b[]); §2.13.7
    public int read(byte b[], int off, int len); §2.13.8
    public void reset(); §2.13.9
    public long skip(long n); §2.13.10
}
```

This class is an abstract class that is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input (I-§2.13.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InputStream.InputStream

public **InputStream()**

The default constructor. This constructor is only called by subclasses.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.available

public int available()
throws IOException

Determines the number of bytes that can be read from this input stream without blocking. The available method of InputStream returns 0. This method should be overridden by subclasses.

Returns:

the number of bytes that can be read from this input stream without blocking.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.close

public void close()
throws IOException

Closes this input stream and releases any system resources associated with the stream.

The close method of InputStream does nothing.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

InputStream.mark

public void mark(int readlimit)

Marks the current position in this input stream. A subsequent call to the reset method (I-§2.13.9) repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The readlimit arguments tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

The mark method of InputStream does nothing.

Parameters:

readlimit- the maximum limit of bytes that can be read before the mark position becomes invalid

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.markSupported

public boolean markSupported()

Determines if this input stream supports the mark ([I-§2.13.4](#)) and reset ([I-§2.13.9](#)) methods. The markSupported method of InputStream returns false.

Returns:

true if this true type supports the mark and reset method; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.read

public abstract int read()
throws IOException

Reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

A subclass must provide an implementation of this method.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public int read(byte b[])
throws IOException

Reads up to b.length bytes of data from this input instream into an array of bytes.

The read method of InputStream calls the the read method of three arguments ([I-§2.13.8](#)) with the arguments b, 0, and b.length.

Parameters:

b- the buffer into which the data is read

Returns:

the total number of bytes read into the buffer, or -1 is there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public int read(byte b[], int off, int len)

throws `IOException`

Reads up to `len` bytes of data from this input stream into an array of bytes. This method blocks until some input is available. If the first argument is null, up to `len` bytes are read and discarded.

The read method of `InputStream` reads a single byte at a time using the read method of zero arguments (I-§2.13.6) to fill in the array. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:

`b`– the buffer into which the data is read

`off`– the start offset of the data

`len`– the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

`IOException` (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.reset

public void reset()
throws IOException

Repositions this stream to the position at the time the mark method (I-§2.13.4) was last called on this input stream.

The reset method of InputStream throws an IOException (I-§2.29), since input streams, by default, do not support mark and reset.

Throws

IOException (I-§2.29)

If this stream has not been marked or if the mark has been invalidated.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InputStream.skip

public long skip(long n)
throws IOException

Skips over and discards n bytes of data from this input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly zero. The actual number of bytes skipped is returned.

The skip method of InputStream creates a byte array of length n and then reads into it until n bytes have been read or the end of the stream has been reached. Subclasses are encouraged to provide a more efficient implementation of this method.

Parameters:

n- the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.14 Class LineNumberInputStream

```
public class java.io.LineNumberInputStream
    extends java.io.FilterInputStream (l-§2.11)
{
    // Constructors
    public LineNumberInputStream(InputStream in); §2.14.1

    // Methods
    public int available(); §2.14.2
    public int getLineNumber(); §2.14.3
    public void mark(int readlimit); §2.14.4
    public int read(); §2.14.5
    public int read(byte b[], int off, int len); §2.14.6
    public void reset(); §2.14.7
    public void setLineNumber(int lineNumber); §2.14.8
    public long skip(long n); §2.14.9
}
```

This class is an input stream filter that provides the added functionality of keeping track of the current line number.

A line is a sequence of bytes ending with either a carriage return character ('\r'), a newline character ('\n'), or a carriage return character followed immediately by a line feed character. In all three cases the line terminating character(s) are returned as a single newline character.

The line number begins at zero, and is incremented by 1 when a read returns a newline character.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.LineNumberInputStream

public LineNumberInputStream(InputStream in)

Constructs a new line number input stream that reads its input from the specified input stream.

Parameters:

`in` - the underlying input stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.available

public int available()
throws IOException

Determines the number of bytes that can be read from this input stream without blocking.

Note that if the underlying input stream ([I-§2.11.1](#)) is able to supply k input characters without blocking, the LineNumberInputStream can guarantee only to provide {ewc msdn cd, EWGraphic, IOS2by 0 /a "sunref.BMP"} characters without blocking, because the k characters from the underlying input stream might consist of {ewc msdn cd, EWGraphic, IOS2by 1 /a "sunref.BMP"} pairs of '\r' and '\n', which are converted to just {ewc msdn cd, EWGraphic, IOS2by 2 /a "sunref.BMP"} '\n' characters.

Returns:

the number of bytes that can be read from this input stream without blocking.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

available in class FilterInputStream ([I-§2.11.3](#)).

{ewl msdn cd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.getLineNumber

public int getLineNumber()

Returns:

the current line number.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.mark

public void mark(int readlimit)

Marks the current position in this input stream. A subsequent call to the reset method (I-§2.14.7) repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The mark method of LineNumberInputStream remembers the current line number in a private variable, and then calls the mark method of the underlying input stream (I-§2.11.1).

Parameters:

readlimit- the maximum limit of bytes that can be read before the mark position becomes invalid

Overrides:

mark in class FilterInputStream (I-§2.11.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.read

public int read()
throws IOException

Reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

The read method of LineNumberInputStream calls the read method of the underlying input stream (I-§2.11.1). It checks for carriage returns and newline characters in the input, and modifies the current line number (I-§2.14.3) as appropriate. A carriage return character or a carriage return followed by a newline character are both converted into a single newline character.

Returns:

the next byte of data, or -1 if the end of this stream is reached

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class FilterInputStream (I-§2.11.7).

public int read(byte b[], int off, int len)
throws IOException

Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

The read method of LineNumberInputStream repeatedly calls the read method of zero arguments (I-§2.14.5) to fill in the byte array.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class FilterInputStream ([I-§2.11.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


LineNumberInputStream.reset

public void reset()
throws IOException

Repositions this stream to the position at the time the mark method (I-§2.14.4) was last called on this input stream.

The reset method of LineNumberInputStream resets the line number to be the line number at the time the mark method was called, and then calls the mark method of the underlying input stream (I-§2.11.1).

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

reset in class FilterInputStream (I-§2.11.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.setLineNumber

public void setLineNumber(int lineNumber)

Sets the line number to the specified argument.

Parameters:

lineNumber- the new line number

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LineNumberInputStream.skip

public long skip(long n)
throws IOException

Skips over and discards n bytes of data from the input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly zero. The actual number of bytes skipped is returned.

The skip method of LineNumberInputStream creates a byte array of length n and then reads into it until n bytes have been read or the end of the stream has been reached.

Parameters:

n- the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

skip in class FilterInputStream (I-§2.11.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.15 Class OutputStream

```
public abstract class java.io.OutputStream
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public OutputStream(); §2.15.1

    // Methods
    public void close(); §2.15.2
    public void flush(); §2.15.3
    public void write(byte b[]); §2.15.4
    public void write(byte b[], int off, int len); §2.15.5
    public abstract void write(int b); §2.15.6
}
```

This class is an abstract class that is the superclass of all classes representing an output stream of bytes.

Applications that need to define a subclass of OutputStream must always provide at least a method that writes one byte of output (I-§2.15.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


OutputStream.OutputStream

public OutputStream()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

OutputStream.close

public void close()
throws IOException

Closes this output stream and releases any system resources associated with this stream.

The close method of OutputStream does nothing.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

OutputStream.flush

public void flush()
throws IOException

Flushes this output stream and forces any buffered output bytes to be written out.

The flush method of OutputStream does nothing.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

OutputStream.write

public void write(byte b[])
throws IOException

Writes b.length bytes from the specified byte array to this output stream.

The write method of OutputStream calls the write method of three arguments (I-§2.15.5) with the three arguments b, 0, and b.length.

Parameters:

b- the data

Throws

IOException (I-§2.29)

If an I/O error occurs.

public void write(byte b[], int off, int len)
throws IOException

Writes len bytes from the specified byte array starting at offset off to this output stream.

The write method of OutputStream calls the write method of one argument on each of the bytes to be written out. Subclasses are encouraged to override this method and provide a more efficient implementation.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException (I-§2.29)

If an I/O error occurs.

public abstract void write(int b)
throws IOException

Writes the specified byte to this output stream.

Subclasses of OutputStream must provide an implementation for this method.

Parameters:

b- the byte

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.16 Class PipedInputStream

```
public class java.io.PipedInputStream
    extends java.io.InputStream (I-§2.13)
{
    // Constructors
    public PipedInputStream(); §2.16.1
    public PipedInputStream(PipedOutputStream src); §2.16.2

    // Methods
    public void close(); §2.16.3
    public void connect(PipedOutputStream src); §2.16.4
    public int read(); §2.16.5
    public int read(byte b[], int off, int len); §2.16.6
}
```

A piped input stream is the receiving end of a communications pipe. Two threads can communicate by having one thread send data through a piped output stream (I-§2.17) and having the other thread read the data through a piped input stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PipedInputStream.PipedInputStream

public PipedInputStream()

Creates a piped input stream that is not yet connected to a piped output stream. It must be connected to a piped output stream, either by the receiver ([I-§2.16.4](#)) or the sender ([I-§2.17.4](#)), before being used.

public PipedInputStream(PipedOutputStream src)
throws IOException

Creates a piped input stream connected to the specified piped output stream.

Parameters:

`src`— the stream to connect to

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PipedInputStream.close

public void close()
throws IOException

Closes this piped input stream and releases any system resources associated with the stream.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

close in class InputStream ([I-§2.13.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PipedInputStream.connect

```
public void connect(PipedOutputStream src)  
throws IOException
```

Connects this piped input stream to a sender.

Parameters:

`src`— the piped output stream to connect to

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PipedInputStream.read

public int read()
throws IOException

Reads the next byte of data from this piped input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because this end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException ([I-§2.29](#))

If the pipe is broken.

Overrides:

read in class InputStream ([I-§2.13.6](#)).

public int read(byte b[], int off, int len)
throws IOException

Reads up to len bytes of data from this piped input stream into an array of bytes. This method blocks until at least one byte of input is available.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.8](#)).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.17 Class PipedOutputStream

```
public class java.io.PipedOutputStream
    extends java.io.OutputStream (I-§2.15)
{
    // Constructors
    public PipedOutputStream(); §2.17.1
    public PipedOutputStream(PipedInputStream snk); §2.17.2

    // Methods
    public void close(); §2.17.3
    public void connect(PipedInputStream snk); §2.17.4
    public void write(byte b[], int off, int len); §2.17.5
    public void write(int b); §2.17.6
}
```

A piped output stream is the sending end of a communications pipe. Two threads can communicate by having one thread send data through a piped output stream (I-§2.16) and having the other thread read the data through a piped input stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PipedOutputStream.PipedOutputStream

public PipedOutputStream()

Creates a piped output stream that is not yet connected to a piped input stream. It must be connected to a piped input stream, either by the receiver (I-§2.16.4) or the sender (I-§2.17.4), before being used.

public PipedOutputStream(PipedInputStream snk)
throws IOException

Creates a piped output stream connected to the specified piped input stream.

Parameters:

snk– the piped input stream to connect to

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PipedOutputStream.close

public void close()
throws IOException

Closes this piped output stream and releases any system resources associated with this stream.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

close in class OutputStream ([I-§2.15.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PipedOutputStream.connect

```
public void connect(PipedInputStream snk)  
throws IOException
```

Connects this piped output stream to a receiver

Parameters:

`snk`– the piped output stream to connect to

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PipedOutputStream.write

```
public void write(byte b[], int off, int len)  
throws IOException
```

Writes len bytes from the specified byte array starting at offset off to this piped output stream.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.5](#)).

```
public void write(int b)  
throws IOException
```

Writes the specified byte to the piped output stream.

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

write in class OutputStream ([I-§2.15.6](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.18 Class PrintStream

```
public class java.io.PrintStream
    extends java.io.FilterOutputStream (l-§2.12)
{
    // Constructors
    public PrintStream(OutputStream out); §2.18.1
    public PrintStream(OutputStream out, boolean autoflush); §2.18.2

    // Methods
    public boolean checkError(); §2.18.3
    public void close(); §2.18.4
    public void flush(); §2.18.5
    public void print(boolean b); §2.18.6
    public void print(char c); §2.18.7
    public void print(char s[]); §2.18.8
    public void print(double d); §2.18.9
    public void print(float f); §2.18.10
    public void print(int i); §2.18.11
    public void print(long l); §2.18.12
    public void print(Object obj); §2.18.13
    public void print(String s); §2.18.14
    public void println(); §2.18.15
    public void println(boolean b); §2.18.16
    public void println(char c); §2.18.17
    public void println(char s[]); §2.18.18
    public void println(double d); §2.18.19
    public void println(float f); §2.18.20
    public void println(int i); §2.18.21
    public void println(long l); §2.18.22
    public void println(Object obj); §2.18.23
    public void println(String s); §2.18.24
    public void write(byte b[], int off, int len); §2.18.25
    public void write(int b); §2.18.26
}
```

A print stream implements an output stream filter that provides convenient methods for printing types other than bytes and arrays of bytes.

In addition, the print stream overrides many of the `InputStream` methods so as not to throw an `IOException`. Instead, an I/O exception causes an internal flag to be set, which the application can check by a call to the `checkError` method (l-§2.18.3).

Only the lower 8 bits of any 16-bit quantity are printed to the stream.

An application can specify at creation time whether a print stream should be flushed every time a newline character is written.

Following are some examples of the use of a print stream:


```
System.out.println("Hello world!");  
System.out.print("x = ");  
System.out.println(x);  
System.out.println("y = " + y);
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PrintStream.PrintStream

public PrintStream(OutputStream out)

Constructs a new print stream that writes its output to the specified underlying output stream (I-§2.12.1).

Parameters:

out- the underlying output stream

public

PrintStream(OutputStream out, boolean autoflush)

Constructs a new print stream that writes its output to the specified underlying output stream (I-§2.12.1). In addition, if the autoflush flag is true, then the underlying output stream's flush method is called any time a newline character is printed.

Parameters:

out- the underlying output stream

autoflush- if true the stream automatically flushes its output when a newline character is printed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PrintStream.checkError

public boolean checkError()

Flushes this print stream's underlying output stream, and returns a boolean indicating if there has been an error on the underlying output stream.

Errors are cumulative; once the print stream has encountered an error, this method will continue to return true on all successive calls.

Returns:

true if the print stream has ever encountered an error on the output stream; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PrintStream.close

public void close()

Closes this print stream and releases any resources associated with the underlying output stream.

The close method of PrintStream calls the close method ([I-§2.15.2](#)) of its underlying output stream ([I-§2.12.1](#)). However, if that close method throws an IOException, this method catches that exception and indicates, instead, that the underlying stream has received an error (see [I-§2.18.3](#)).

Overrides:

close in class FilterOutputStream ([I-§2.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PrintStream.flush

public void flush()

Flushes this print stream. This forces any buffered output bytes to be written to the underlying stream.

The flush method of PrintStream calls the flush method (I-§2.15.3) of its underlying output stream (I-§2.12.1). However, if that flush method throws an IOException, this method catches that exception and indicates, instead, that the underlying stream has received an error (see I-§2.18.3).

Overrides:

flush in class FilterOutputStream (I-§2.12.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PrintStream.print

`public void print(boolean b)`

Prints the string "true" to the underlying output stream ([I-§2.12.1](#)) if the value of the boolean argument is true; otherwise, prints the string "false" to the underlying output stream.

Parameters:

b- a boolean to be printed

`public void print(char c)`

Prints the low eight bits of the character argument to this print stream's underlying output stream ([I-§2.12.1](#)).

Parameters:

c- a char to be printed

`public void print(char s[])`

Prints the low eight bits of each of the characters in the character array to this print stream's underlying output stream ([I-§2.12.1](#)).

Parameters:

s- an array of chars to be printed

`public void print(double d)`

Prints the string representation of the double to this print stream's underlying output stream ([I-§2.12.1](#)). The string representation is identical to the one returned by the toString method ([I-§1.6.21](#)) of class Double with the argument d.

Parameters:

d- a double to be printed

public void print(float f)

Prints the string representation of the float to this print stream's underlying output stream (I-§2.12.1). The string representation is identical to the one returned by the toString method (I-§1.7.22) of class Float with the argument f.

Parameters:

f- a float to be printed

public void print(int i)

Prints the string representation of the int to this print stream's underlying output stream (I-§2.12.1). The string representation is identical to the one returned by the toString method (I-§1.8.20) of class Integer with the argument i.

Parameters:

i- an int to be printed

public void print(long l)

Prints the string representation of the long to this print stream's underlying output stream (I-§2.12.1). The string representation is identical to the one returned by the toString method (I-§1.9.20) of class Long with the argument l.

Parameters:

l- a long to be printed.

public void print(Object obj)

Prints the string representation of the Object to this print stream's underlying output stream (I-§2.12.1). The string representation is identical to the one returned by calling the Object argument's toString method (I-§1.12.9).

Parameters:

obj – an Object to be printed

public void print(String s)

If the string argument is null, the string "null" is printed to this print stream's underlying output stream. Otherwise, the low eight bits of each of the characters in the string is printed to the underlying output stream (I-§2.12.1).

Parameters:

s – a String to be printed

{ewl msdncl.dll, ewcright, /c"Microsoft"}

PrintStream.println

public void println()

Prints a newline character to this print stream's underlying output stream (I-§2.12.1).

public void println(boolean b)

Prints the string "true" followed by a newline character to this print stream's underlying output stream if the value of the boolean argument is true; otherwise, prints the string "false" followed by a newline character to the underlying output stream (I-§2.12.1).

Parameters:

b- a boolean to be printed

public void println(char c)

Prints the low eight bits of the character argument followed by a newline character to this print stream's underlying output stream (I-§2.12.1).

Parameters:

c- a char to be printed

public void println(char s[])

Prints the low eight bits of each of the characters in the character array, followed by a newline character, to this print stream's underlying output stream (I-§2.12.1).

Parameters:

s- an array of characters to be printed

public void println(double d)

Prints the string representation of the double followed by a newline to this print stream's

underlying output stream ([I-§2.12.1](#)). The string representation is identical to the one returned by the toString [method](#) ([I-§1.6.21](#)) of class Double with the [argument](#) d.

Parameters:

d- a double to be printed

public void println(float f)

Prints the string representation of the float followed by a newline to this print stream's underlying output stream ([I-§2.12.1](#)). The string representation is identical to the one returned by the toString [method](#) ([I-§1.8.20](#)) of class Integer with the [argument](#) f.

Parameters:

f- a float to be printed

public void println(int i)

Prints the string representation of the int followed by a newline to this print stream's underlying output stream ([I-§2.12.1](#)). The string representation is identical to the one returned by the toString [method](#) ([I-§1.8.20](#)) of class Integer with the [argument](#) i.

Parameters:

i- an int to be printed

public void println(long l)

Prints the string representation of the long followed by a newline to this print stream's underlying output stream ([I-§2.12.1](#)). The string representation is identical to the one returned by the toString [method](#) ([I-§1.9.20](#)) of class Long with the [argument](#) l.

Parameters:

l- a long to be printed

public void println(Object obj)

Prints the string representation of the Object followed by a newline to this print stream's underlying output stream (I-§2.12.1). The string representation is identical to the one returned by calling the Object argument's toString method (I-§1.12.9).

Parameters:

obj- an Object to be printed

public void println(String s)

If the string argument is null, the string "null" followed by a newline character is printed to this print stream's underlying output stream. Otherwise, the low eight bits of each of the characters in the string, followed by a newline character, is printed to the underlying output stream (I-§2.12.1).

Parameters:

s- a String to be printed

{ewl msdncl.dll, ewcright, /c"Microsoft"}

PrintStream.write

public void write(byte b[], int off, int len)

Writes len bytes from the specified byte array starting at offset off to this print stream's underlying output stream (I-§2.12.1).

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Overrides:

write in class FilterOutputStream (I-§2.12.6).

public void write(int b)

Writes the specified byte to this print stream.

The write method of PrintStream calls the write method of its underlying stream (I-§2.12.1). In addition, if the character is a newline character and autoflush is turned on, then the print stream's flush method (I-§2.18.5) is called.

If any IOException is thrown while writing the byte, the exception is caught, and instead an internal error flag is set; the value of the flag can be checked by a call to the checkError method (I-§2.18.3).

Parameters:

b- the byte

Overrides:

write in class FilterOutputStream (I-§2.12.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.19 Class PushbackInputStream

```
public class java.io.PushbackInputStream
    extends java.io.FilterInputStream (l-§2.11)
{
    // Fields
    protected int pushBack; §2.19.1

    // Constructors
    public PushbackInputStream(InputStream in); §2.19.2

    // Methods
    public int available(); §2.19.3
    public boolean markSupported(); §2.19.4
    public int read(); §2.19.5
    public int read(byte bytes[], int offset, int length); §2.19.6
    public void unread(int ch); §2.19.7
}
```

This class is an input stream filter that provides a one-byte push back buffer. This feature allows an application to "unread" the last character that it read. The next time that a read is performed on the input stream filter, the "unread" character is re-read.

This functionality is useful in situations where it is useful for a fragment of code to read an indefinite number of data bytes that are delimited by particular byte values; after reading the terminating byte, the code fragment can "unread" it, so that the next read operation on the input stream will re-read the byte that was pushed back.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PushbackInputStream.pushBack

protected `int pushBack`

A character that has been "unread" and that will be the next byte read. The value -1 indicates no character in the buffer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PushbackInputStream.PushbackInputStream

public PushbackInputStream(InputStream in)

Constructs a new push back input stream that reads its input from the specified input stream.

Parameters:

`in`– the underlying input stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PushbackInputStream.available

public int available()
throws IOException

Determines the number of bytes that can be read from this input stream without blocking.

The available method of PushbackInputStream calls the available method of its underlying input stream (I-§2.11.1); it returns that value if there is no character that has been pushed back, or that value plus one if there is a character that has been pushed back.

Returns:

the number of bytes that can be read from the input stream without blocking.

Overrides:

available in class FilterInputStream (I-§2.11.3).

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PushbackInputStream.markSupported

public boolean markSupported()

Determines if the input stream supports the mark ([I-§2.13.4](#)) and reset ([I-§2.13.9](#)) methods. The markSupported method of PushbackInputStream always returns false.

Returns:

true if this stream type supports the mark and reset methods; false otherwise.

Overrides:

markSupported in class FilterInputStream ([I-§2.11.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PushbackInputStream.read

public int read()
throws IOException

Reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

The read method of PushbackInputStream returns the just pushed-back character, if there is one, and otherwise calls the the read method of its underlying input stream (I-§2.13.6) and returns whatever value that method returns.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class FilterInputStream (I-§2.11.7).

public int read(byte bytes[], int offset, int length)
throws IOException

Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until at least one byte of input is available.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class FilterInputStream (I-§2.11.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PushbackInputStream.unread

public void unread(int ch)
throws IOException

Pushes back a character so that it is read again by the next call to the read method on this input stream.

Parameters:

ch- the character to push back

Throws

IOException ([I-§2.29](#))

If the application attempts to push back a character before the previously pushed back character has been read.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.20 Class RandomAccessFile

```
public class java.io.RandomAccessFile
    extends java.lang.Object (l-§1.12)
    implements java.io.DataOutput (l-§2.25),
               java.io.DataInput (l-§2.24)
{
    // Constructors
    public RandomAccessFile(File file, String mode); §2.20.1
    public RandomAccessFile(String name, String mode); §2.20.2

    // Methods
    public void close(); §2.20.3
    public final FileDescriptor getFD(); §2.20.4
    public long getFilePointer(); §2.20.5
    public long length(); §2.20.6
    public int read(); §2.20.7
    public int read(byte b[]); §2.20.8
    public int read(byte b[], int off, int len); §2.20.9
    public final boolean readBoolean(); §2.20.10
    public final byte readByte(); §2.20.11
    public final char readChar(); §2.20.12
    public final double readDouble(); §2.20.13
    public final float readFloat(); §2.20.14
    public final void readFully(byte b[]); §2.20.15
    public final void readFully(byte b[], int off, int len); §2.20.16
    public final int readInt(); §2.20.17
    public final String readLine(); §2.20.18
    public final long readLong(); §2.20.19
    public final short readShort(); §2.20.20
    public final int readUnsignedByte(); §2.20.21
    public final int readUnsignedShort(); §2.20.22
    public final String readUTF(); §2.20.23
    public void seek(long pos); §2.20.24
    public int skipBytes(int n); §2.20.25
    public void write(byte b[]); §2.20.26
    public void write(byte b[], int off, int len); §2.20.27
    public void write(int b); §2.20.28
    public final void writeBoolean(boolean v); §2.20.29
    public final void writeByte(int v); §2.20.30
    public final void writeBytes(String s); §2.20.31
    public final void writeChar(int v); §2.20.32
    public final void writeChars(String s); §2.20.33
    public final void writeDouble(double v); §2.20.34
    public final void writeFloat(float v); §2.20.35
```



```
public final void writeInt(int v); §2.20.36  
public final void writeLong(long v); §2.20.37  
public final void writeShort(int v); §2.20.38  
public final void writeUTF(String str); §2.20.39  
}
```

Instances of this class support both reading and writing to a random access file. An application can modify the position in the file at which the next read or write occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.RandomAccessFile

public RandomAccessFile(File file, String mode)
throws IOException

Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.

The mode argument must either be equal to "r" or to "rw," indicating either to open the file for input, or for both input and output, respectively.

Parameters:

file- the file object

mode- the access mode

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Throws

IllegalArgumentException ([I-§1.32](#))

If the mode argument is not equal to "r" or to "rw."

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its checkRead method ([I-§1.15.19](#)) is called with the path name ([I-§2.7.16](#)) of the File argument to see if the application is allowed read access to the file. If the mode argument is equal to "rw", its checkWrite method ([I-§1.15.19](#)) is also called with the path name to see if the application is allowed write access to the file.

public RandomAccessFile(String name, String mode)
throws IOException

Creates an random access file stream to read from, and optionally to write to, a file with the specified name.

The mode argument must either be equal to "r" or "rw," indicating either to open the file for input or for both input and output.

Parameters:

`name`– the system dependent file name

`mode`– the access mode

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Throws

IllegalArgumentException ([I-§1.32](#))

If the mode argument is not equal to "r" or to "rw."

Throws

SecurityException ([I-§1.43](#))

If a security manager exists, its `checkRead` [method \(I-§1.15.19\)](#) is called with the name argument to see if the application is allowed read access to the file. If the mode argument is equal to "rw," its `checkWrite` [method \(I-§1.15.19\)](#) is also called with the name argument to see if the application is allowed write access to the file. Either of these may result in a security exception (I-§1.43).

{`ewl msdncd.dll, ewcright, /c"Microsoft"`}

RandomAccessFile.close

public void close()
throws IOException

Closes this random access file stream and releases any system resources associated with the stream.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.getFD

```
public final FileDescriptor getFD()  
throws IOException
```

Returns:

the file descriptor object (I-§2.8) associated with this stream.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.getFilePointer

```
public long getFilePointer()  
throws IOException
```

Returns the current offset in this file.

Returns:

the offset from the beginning of the file, in bytes, at which the next read or write occurs.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.length

public long length()
throws IOException

Returns:

the length of this file.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.read

```
public int read()  
throws IOException
```

Reads a byte of data from this file. This method blocks if no input is yet available.

Returns:

the next byte of data, or -1 if the end of the file is reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
public int read(byte b[])  
throws IOException
```

Reads up to b.length bytes of data from this file into an array of bytes. This method blocks until at least one byte of input is available.

Parameters:

b- the buffer into which the data is read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of this file has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
public int read(byte b[], int off, int len)  
throws IOException
```

Reads up to len bytes of data from this file into an array of bytes. This method blocks until at least one byte of input is available.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

`len` - the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the file has been reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.readBoolean

public final boolean readBoolean()
throws IOException

Reads a boolean from this file. This method reads a single byte from the file. A value of 0 represents false. Any other value represents true. This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the boolean value read.

Throws

EOFException ([I-§2.24](#))

If this file has reached the end.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readByte

public final byte readByte()
throws IOException

Reads a signed 8-bit value from this file. This method reads a byte from the file. If the byte read is b, where 0 is less than or equal to b is less than or equal to 255, then the result is:

(byte) (b)

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of this file as a signed 8-bit byte.

Throws

EOFException (I-§2.24)

If this file has reached the end.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readChar

public final char readChar()
throws IOException

Reads a Unicode character from this file. This method reads two bytes from the file. If the bytes read, in order, are b1 and b2, where 0 is less than or equal to b1, b2 is less than or equal to 255, then the result is equal to:

$$(\text{char}) ((b1 \ll 8) | b2)$$

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this file as an Unicode character.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readDouble

public final double readDouble()
throws IOException

Reads a double from this file. This method reads a long value as if by the readLong method (I-§2.20.19) and then converts that long to a double using the longBitsToDouble method (I-§1.6.18) in class Double.

This method blocks until either the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next eight bytes of this file, interpreted as a double.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading eight bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readFloat

public final float readFloat()
throws IOException

Reads a float from this file. This method reads an int value as if by the readInt method (I-§2.20.17) and then converts that int to a float using the intBitsToFloat method (I-§1.7.14) in class Float.

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next four bytes of this file, interpreted as a float.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading four bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readFully

```
public final void readFully(byte b[])  
throws IOException
```

Reads b.length bytes from this file into the byte array. This method reads repeatedly from the file until all the bytes are read. This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

Parameters:

b- the buffer into which the data is read

Throws

EOFException ([I-§2.24](#))

If this file reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
public final void readFully(byte b[], int off, int len)  
throws IOException
```

Reads exactly len bytes from this file into the byte array. This method reads repeatedly from the file until all the bytes are read. This method blocks until either all the bytes are read, the end of the stream is detected, or an exception is thrown.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the number of bytes to read read

Throws

EOFException ([I-§2.24](#))

If this file reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readInt

public final int readInt()
throws IOException

Reads a signed 32-bit integer from this file. This method reads four bytes from the file. If the bytes read, in order, are b1, b2, b3, and b4 where 0 is less than or equal to b1, b2, b3, b4 is less than or equal to 255, then the result is equal to

$$(b1 \ll 24) \mid (b2 \ll 16) + (b3 \ll 8) + b4$$

This method blocks until the four bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next four bytes of this file, interpreted as an int.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading four bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readLine

public **final** **String** **readLine()**
throws **IOException**

Reads the next line of text from this file. This method successively reads bytes from the file until it reaches the end of a line of text.

A line of text is terminated by a carriage return character ('\r'), a newline character ('\n'), a carriage return character immediately followed by a newline character, or the end of the input stream. The line-terminating character(s), if any, are included as part of the string returned.

This method blocks until a newline character is read, a carriage return and the byte following it are read (to see if it is a newline), the end of the stream is detected, or an exception is thrown.

Returns:

the next line of text from this file.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readLong

public final long readLong()
throws IOException

Reads a signed 64-bit integer from this file. This method reads eight bytes from the file. If the bytes read, in order, are b1, b2, b3, b4, b5, b6, b7, and b8, where:

0 is less than or equal to b1, b2, b3, b4, b5, b6, b7, b8 is less than or equal to 255

then the result is equal to:

$$\begin{aligned} & ((\text{long})b1 \ll 56) + ((\text{long})b2 \ll 48) \\ & + ((\text{long})b3 \ll 40) + ((\text{long})b4 \ll 32) \\ & + ((\text{long})b5 \ll 24) + ((\text{long})b6 \ll 16) \\ & + ((\text{long})b7 \ll 8) + b8 \end{aligned}$$

This method blocks until the eight bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next eight bytes of this file, interpreted as a long.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading eight bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readShort

public **final** **short** readShort()
throws IOException

Reads a signed 16-bit number from this file. The method reads two bytes from this file. If the two bytes read, in order, are b1 and b2, where each of the two values is between 0 and 255, inclusive, then the result is equal to:

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this file, interpreted as a signed 16-bit number.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readUnsignedByte

public final int readUnsignedByte()
throws IOException

Reads an unsigned 8-bit number from this file. This method reads a byte from this file and returns that byte.

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Returns:

the next byte of this file, interpreted as an unsigned 8-bit number.

Throws

EOFException (I-§2.24)

If this file has reached the end.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readUnsignedShort

public final int readUnsignedShort()
throws IOException

Reads an unsigned 16-bit number from this file. This method reads two bytes from the file. If the bytes read, in order, are b1 and b2, where 0 is less than or equal to b1, b2 is less than or equal to 255, then the result is equal to:

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

the next two bytes of this file, interpreted as an unsigned 16-bit integer.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading two bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.readUTF

public final String readUTF()
throws IOException

Reads in a string from this file. The string has been encoded using a modified UTF-8 format.

The first two bytes from are read as if by readUnsignedShort (I-§2.20.22). This value gives the number of following bytes that are in the encoded string. (Note: *not* the length of the resulting string). The following bytes are then interpreted as bytes encoding characters in the UTF-8 format (page I-225) and are converted into characters.

This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

Returns:

a Unicode string.

Throws

EOFException (I-§2.24)

If this file reaches the end before reading all the bytes.

Throws

UTFDataFormatException (I-§2.31)

If the bytes do not represent a valid UTF-8 encoding of a Unicode string.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.seek

public void seek(long pos)
throws IOException

Sets the offset from the beginning of this file at which the next read or write occurs.

Parameters:

`pos` - the absolute position

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.skipBytes

```
public int skipBytes(int n)  
throws IOException
```

Skips exactly n bytes of input.

This method blocks until all the bytes are skipped, the end of the stream is detected, or an exception is thrown.

Parameters:

n- the number of bytes to be skipped

Returns:

the number of bytes skipped, which is always n.

Throws

EOFException ([I-§2.24](#))

If this file reaches the end before skipping all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.write

```
public void write(byte b[])  
throws IOException
```

Writes b.length bytes from the specified byte array starting at offset off to this file.

Parameters:

b- the data

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
public void write(byte b[], int off, int len)  
throws IOException
```

Writes len bytes from the specified byte array starting at offset off to this file.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
public void write(int b)  
throws IOException
```

Writes the specified byte to this file

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeBoolean

```
public final void writeBoolean(boolean v)  
throws IOException
```

Writes a boolean to the file as a one-byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0.

Parameters:

`v` – a boolean value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.writeByte

```
public final void writeByte(int v)  
throws IOException
```

Writes out a byte to the file as a one-byte value.

Parameters:

v – a byte value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeBytes

public final void writeBytes(String s)
throws IOException

Writes out the string to the file as a sequence of bytes. Each character in the string is written out, in sequence, by discarding its high eight bits.

Parameters:

s – a string of bytes to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.writeChar

```
public final void writeChar(int v)  
throws IOException
```

Writes a char to the file as a two-byte value, high byte first.

Parameters:

v – a char value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeChars

public final void writeChars(String s)
throws IOException

Writes a string to the file as a sequence of characters. Each character is written to the data output stream as if by the writeChar [method \(I-§2.20.32\)](#).

Parameters:

s – a String value to be written

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.writeDouble

public final void writeDouble(double v)
throws IOException

Converts the double argument to a long using the doubleToLongBits method (I-§1.6.8) in class Double, and then writes that long value to the file as an eight-byte quantity, high-byte first.

Parameters:

v— a double value to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RandomAccessFile.writeFloat

```
public final void writeFloat(float v)  
throws IOException
```

Converts the float argument to an int using the floatToIntBits method (I-§1.7.11) in class Float, and then writes that int value to the file as a four-byte quantity, high-byte first.

Parameters:

v – a float value to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeInt

```
public final void writeInt(int v)  
throws IOException
```

Writes an int to the file as four bytes, high-byte first.

Parameters:

v— an int to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeLong

```
public final void writeLong(long v)  
throws IOException
```

Writes a long to the file as eight bytes, high-byte first.

Parameters:

v— a long to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeShort

```
public final void writeShort(int v)  
throws IOException
```

Writes a short to the file as two bytes, high-byte first.

Parameters:

v – a short to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RandomAccessFile.writeUTF

public final void writeUTF(String str)
throws IOException

Writes out a string to the file using UTF-8 encoding in a machine-independent manner.

First, two bytes are written to the file as if by the writeShort [method \(I-§2.20.38\)](#) giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the UTF-8 encoding ([page I-225](#)) for each character.

Parameters:

`str` - a string to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.21 Class SequenceInputStream

```
public class java.io.SequenceInputStream
    extends java.io.InputStream (l-§2.13)
{
    // Constructors
    public SequenceInputStream(Enumeration e); §2.21.1
    public SequenceInputStream(InputStream s1, InputStream s2); §2.21.2

    // Methods
    public void close(); §2.21.3
    public int read(); §2.21.4
    public int read(byte buf[], int pos, int len); §2.21.5
}
```

The sequence input stream class allows an application to combine several input streams serially and make them appear as if they were a single input stream. Each input stream is read from, in turn, until it reaches the end of the stream. The sequence input stream class then closes that stream and automatically switches to the next input stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SequenceInputStream.SequenceInputStream

public SequenceInputStream(Enumeration e)

Constructs a new sequence input stream initialized to the specified enumeration (I-§3.11) of input streams. Each object in the enumeration must be an InputStream.

Parameters:

e – an enumeration of input streams

public SequenceInputStream(InputStream s1, InputStream s2)

Constructs a new sequence input stream initialized to read first from the input stream s1, and then from the input stream s2.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SequenceInputStream.close

public void close()
throws IOException

Closes this input stream and releases any system resources associated with the stream.

The close method of SequenceInputStream calls the close method of both the substream from which it is currently reading and the close method of all the substreams that it has not yet begun to read from.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

close in class InputStream ([I-§2.13.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SequenceInputStream.read

public int read()
throws IOException

Reads the next byte of data from this input stream. The byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

The read method of SequenceInputStream tries to read one character from the current substream. If it reaches the end of the stream, it calls the close method of the current substream and begins reading from the next substream.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

Overrides:

read in class InputStream ([I-§2.13.6](#)).

public int read(byte buf[], int pos, int len)
throws IOException

Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until at least one byte of input is available. If the first argument is null, up to len bytes are read and discarded.

The read method of SequenceInputStream tries to read the data from the current substream. If it fails to read any characters because the substream has reached the end of the stream, it calls the close method of the current substream and begins reading from the next substream.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Throws

IOException (I-§2.29)

If an I/O error occurs.

Overrides:

read in class InputStream (I-§2.13.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.22 Class StreamTokenizer

```
public class java.io.StreamTokenizer
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public double nval; §2.22.1
    public String sval; §2.22.2
    public int ttype; §2.22.3

    // possible values for the ttype field
    public final static int TT_EOF; §2.22.4
    public final static int TT_EOL; §2.22.5
    public final static int TT_NUMBER; §2.22.6
    public final static int TT_WORD; §2.22.7

    // Constructors
    public StreamTokenizer(InputStream I); §2.22.8

    // Methods
    public void commentChar(int ch); §2.22.9
    public void eolIsSignificant(boolean flag); §2.22.10
    public int lineno(); §2.22.11
    public void lowerCaseMode(boolean fl); §2.22.12
    public int nextToken(); §2.22.13
    public void ordinaryChar(int ch); §2.22.14
    public void ordinaryChars(int low, int hi); §2.22.15
    public void parseNumbers(); §2.22.16
    public void pushBack(); §2.22.17
    public void quoteChar(int ch); §2.22.18
    public void resetSyntax(); §2.22.19
    public void whitespaceChars(int low, int hi); §2.22.20
    public void slashStarComments(boolean flag); §2.22.21
    public String toString(); §2.22.22
    public void whitespaceChars(int low, int hi); §2.22.23
    public void wordChars(int low, int hi); §2.22.24
}
```

The StreamTokenizer class takes an input stream and parses it into "tokens," allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.

Each byte read from the input stream is regarded as a character in the range 'u0000' through 'u00FF'. The character value is used to look up five possible attributes of the character: *whitespace*, *alphabetic*, *numeric*, *string quote*, and *comment character*. Each character can have zero or more of these attributes.

In addition an instance has four flags. These flags indicate:

- Whether line terminators are to be returned as tokens or treated as whitespace that merely separates tokens.
- Whether C-style comments are to be recognized and skipped.
- Whether C++-style comments are to be recognized and skipped.
- Whether the characters of identifiers are converted to lowercase.

A typical application first constructs an instance of this class, sets up the syntax tables, and then repeatedly loops calling the nextToken method (I-§2.22.13) in each iteration of the loop until it returns the value TT_EOF (I-§2.22.4).

```
{ewl msdncl.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.nval

public double nval

If the current token is a number, this field contains the value of that number. The current token is a number when the value of the ttype field (I-§2.22.3) is TT_NUMBER (I-§2.22.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.sval

public String sval

If the current token is a word token, this field contains a string giving the characters of the word token. When the current token is a quoted string token, this field contains the body of the string.

The current token is a word when the value of the ttype field ([I-§2.22.3](#)) is TT_WORD ([I-§2.22.7](#)). The current token is a quoted string token when the value of the ttype field is a quote character (see [I-§2.22.18](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.ttype

`public int ttype`

After a call to the `nextToken` [method \(I-§2.22.13\)](#), this field contains the type of the token just read. For a single character token, its value is the single character, converted to an integer. For a quoted string token (see [I-§2.22.18](#)), its value is the quote character. Otherwise, its value is one of the following:

- `TT_WORD` ([I-§2.22.7](#)) indicates that the token is a word.
- `TT_NUMBER` ([I-§2.22.6](#)) indicates that the token is a number.
- `TT_EOL` ([I-§2.22.5](#)) indicates that the end of line has been read. The field can only have this value if the `eolIsSignificant` [method \(I-§2.22.10\)](#) has been called with the [argument](#) `true`.
- `TT_EOF` ([I-§2.22.4](#)) indicates that the end of the input stream has been reached.

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

StreamTokenizer.TT_EOF

```
public final static int TT_EOF = -1
```

A constant indicating that the end of the stream has been read.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.TT_EOL

```
public final static int TT_EOL = '\n'
```

A constant indicating that the end of line has been read.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.TT_NUMBER

```
public final static int TT_NUMBER = -2
```

A constant indicating that a number has been read.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.TT_WORD

```
public final static int TT_WORD = -3
```

A constant indicating that a token has been read.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.StreamTokenizer

public StreamTokenizer(InputStream I)

Creates a stream tokenizer that parses the specified input stream. The stream tokenizer is initialized to the following default state:

- All byte values 'A' through 'Z', 'a' through 'z', and '\u00A0' through '\u00FF' are considered to be alphabetic.
- All byte values '\u0000' through '\u0020' are considered to be whitespace.
- '/' is a comment character.
- Single quote '"' and double quote '"' are string quote characters.
- Numbers are parsed.
- End of lines are not treated as whitespace, not as separate tokens.
- C-style and C++-style comments are not recognized.

Parameters:

I – the input stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.commentChar

public void commentChar(int ch)

Specified that the character argument starts a single line comment. All characters from the comment character to the end of the line are ignored by this stream tokenizer.

Parameters:

ch- the character

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StreamTokenizer.eolIsSignificant

public void eolIsSignificant(boolean flag)

If the **flag argument** is true, this tokenizer treats end of lines as **tokens**; the `nextToken` method (I-§2.22.13) returns `TT_EOL` (I-§2.22.5) and also sets the `ttype` field (I-§2.22.3) to this value when an end-of-line is read.

A line is a sequence of characters ending with either a **carriage return** character (`'\r'`) or a newline character (`'\n'`). In addition, a **carriage return** character followed immediately by a newline character is treated as a single end-of-line token.

If the **flag** is false, end of line characters are treated as whitespace and serve only to separate **tokens**.

Parameters:

flag– true indicates that end-of-line characters are separate **tokens**; false indicates that end-of-line characters are whitespace

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.lineno

public int lineno()

Returns:

the current line number of this stream tokenizer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.lowerCaseMode

public void lowerCaseMode(boolean fl)

If the flag argument is true, then the value in the sval field is lowercased whenever a word token is returned (the ttype field ([I-§2.22.3](#)) has the value TT_WORD ([I-§2.22.7](#)) by the nextToken method ([I-§2.22.13](#)) of this tokenizer.

If the flag argument is false, then the sval field is not modified.

Parameters:

fl- true indicates that all word tokens should be lowercase

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.nextToken

public int nextToken()
throws IOException

Parses the next token from the input stream of this tokenizer. The type of the next token is returned in the ttype field ([I-§2.22.3](#)). Additional information about the token may be in the nval field ([I-§2.22.1](#)) or the sval field ([I-§2.22.2](#)) of this tokenizer.

Returns:

The value of the ttype field ([I-§2.22.3](#)).

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.ordinaryChar

public void ordinaryChar(int ch)

Specifies that the character argument is "ordinary" in this tokenizer. It removes any special significance the character has as a comment character, word component, string delimiter, whitespace, or number character. When such a character is encountered by the parser, the parser treats it as a single-character token and sets ttype field (I-§2.22.3) to the character value.

Parameters:

ch- the character

{ewl msdncl.dll, ewcright, /c"Microsoft"}

StreamTokenizer.ordinaryChars

public void ordinaryChars(int low, int hi)

Specifies that all characters *c* in the range *low* is less than or equal to *c* is less than or equal to *high* are "ordinary" in this tokenizer. See the ordinaryChar method (I-§2.22.14) for more information on a character being ordinary.

Parameters:

low- the low end of the range

hi- the high end of the range

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.parseNumbers

public void parseNumbers()

Specifies that numbers should be parsed by this tokenizer. The syntax table of this tokenizer is modified so that each of the twelve characters.

0 1 2 3 4 5 6 7 8 9 . -

has the "numeric" attribute.

When the parser encounters a word token that has the format of a double precision floating point number, it treats the token as a number rather than a word, by setting the the ttype field (I-§2.22.3) to the value TT_NUMBER (I-§2.22.6) and putting the numeric value of the token into the nval field (I-§2.22.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.pushBack

public void pushBack()

Causes the next call to the nextToken method (I-§2.22.13) of this tokenizer to return the current value in the ttype field (I-§2.22.3), and not to modify the value in the nval (I-§2.22.1) or sval (I-§2.22.2) field.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.quoteChar

public void quoteChar(int ch)

Specifies that matching pairs of this character delimit string constants in this tokenizer.

When the nextToken method (I-§2.22.13) encounters a string constant, the ttype field (I-§2.22.3) is set to the string delimiter and the sval field (I-§2.22.2) is set to the body of the string.

If a string quote character is encountered, then a string is recognized, consisting of all characters after (but not including) the string quote character, up to (but not including) the next occurrence of that same string quote character, or a line terminator, or end-of-file. The usual escape sequences such as "\n" and "\t" are recognized and converted to single characters as the string is parsed.

Parameters:

ch- the character

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.resetSyntax

public void resetSyntax()

Resets this tokenizer's syntax table so that all characters are "ordinary." See the ordinaryChar method (I-§2.22.14) for more information on a character being ordinary.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.slashSlashComments

public void slashSlashComments (boolean flag)

If the flag argument is true, this stream tokenizer recognizes C++ style comments. Any occurrence of two consecutive slash characters ("/") is treated as the beginning of a comment that extends to the end of the line.

If the flag argument is false, then C++ style comments are not treated specially.

Parameters:

flag- true indicates to recognize and ignore C++ style comments

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.slashStarComments

public void slashStarComments(boolean flag)

If the flag argument is true, this stream tokenizer recognizes C style comments. All text between successive occurrences of /* and */ are discarded.

If the flag argument is false, then C style comments are not treated specially.

Parameters:

flag- true indicates to recognize and ignore C style comments

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.toString

public String toString()

Returns the string representation of the current stream token.

Returns:

a string representation of the token specified by the ttype ([I-§2.22.3](#)), nval ([I-§2.22.1](#)), and sval ([I-§2.22.2](#)) fields.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.whitespaceChars

public void whitespaceChars(int low, int hi)

Specifies that all characters *c* in the range *low* is less than or equal to *c* is less than or equal to *high* are whitespace characters. Whitespace characters serve only to separate tokens in the input stream.

Parameters:

low- the low end of the range

hi- the high end of the range

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StreamTokenizer.wordChars

public void wordChars(int low, int hi)

Specifies that all characters *c* in the range *low* is less than or equal to *c* is less than or equal to *high* are word constituents. A word token consists of a word constituent followed by zero or more word constituents or number constituents.

Parameters:

low- the low end of the range

hi- the high end of the range

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.23 Class StringBufferInputStream

```
public class java.io.StringBufferInputStream
    extends java.io.InputStream (l-§2.13)
{
    // Fields
    protected String buffer; §2.23.1
    protected int count; §2.23.2
    protected int pos; §2.23.3

    // Constructors
    public StringBufferInputStream(String s); §2.23.4

    // Methods
    public int available(); §2.23.5
    public int read(); §2.23.6
    public int read(byte b[], int off, int len); §2.23.7
    public void reset(); §2.23.8
    public long skip(long n); §2.23.9
}
```

This class allows an application to create an input stream in which the bytes read are supplied by the contents of a string. Applications can also read bytes from a byte array by using a ByteArrayInputStream (l-§2.3).

Only the low eight bits of each character in the string are used by this class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBufferInputStream.buffer

protected String buffer

The string from which bytes are read.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.count

protected int count

The number of valid characters in the input stream buffer (I-§2.23.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.pos

protected int pos

The index of the next character to read from the input stream buffer (I-§2.23.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.StringBufferInputStream

public StringBufferInputStream(String s)

Creates a string input stream to read data from the specified string.

Parameters:

s – the underlying input buffer

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringBufferInputStream.available

public `int available()`

Determines the number of bytes that can be read from the input stream without blocking.

Returns:

the value of `count - pos`, which is the number of bytes remaining to be read from the input buffer.

Overrides:

available in class `InputStream` ([I-§2.13.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.read

public int read()

Reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.

The read method of StringBufferInputStream cannot block. It returns the low 8 bits of the next character in this input stream's buffer.

Returns:

the next byte of data, or -1 if the end of the stream is reached.

Overrides:

read in class InputStream (I-§2.13.6).

public int read(byte b[], int off, int len)

Reads up to len bytes of data from this input stream into an array of bytes.

The read method of StringBufferInputStream cannot block. It copies the low 8-bits from the characters in this input stream's buffer into the byte array argument.

Parameters:

b- the buffer into which the data is read

off- the start offset of the data

len- the maximum number of bytes read

Returns:

the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Overrides:

read in class InputStream (I-§2.13.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.reset

public void reset()

Resets the input stream to begin reading from the first character of this input stream's underlying buffer.

Overrides:

reset in class InputStream ([I-§2.13.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringBufferInputStream.skip

public long skip(long n)

Skips n bytes of input from this input stream. Fewer bytes might be skipped if the end of the input stream is reached.

Parameters:

n- the number of bytes to be skipped

Returns:

the actual number of bytes skipped.

Overrides:

skip in class InputStream ([I-§2.13.10](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.24 Interface DataInput

```
public interface java.io.DataInput
{
    // Methods
    public abstract boolean readBoolean(); §
    public abstract byte readByte(); §2.24.2
    public abstract char readChar(); §2.24.3
    public abstract double readDouble(); §2.24.4
    public abstract float readFloat(); §2.24.5
    public abstract void readFully(byte b[]); §2.24.6
    public abstract void §2.24.7
        readFully(byte b[], int off, int len);
    public abstract int readInt(); §2.24.8
    public abstract String readLine(); §2.24.9
    public abstract long readLong(); §2.24.10
    public abstract short readShort(); §2.24.11
    public abstract int readUnsignedByte(); §2.24.12
    public abstract int readUnsignedShort(); §2.24.13
    public abstract String readUTF(); §2.24.14
    public abstract int skipBytes(int n); §2.24.15
}
```

The data input interface is implemented by streams that can read primitive Java data types from a stream in a machine-independent manner.

See Also:

DataInputStream (I-§2.5)

DataOutput (I-§2.25).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readBoolean

```
public abstract boolean readBoolean()  
throws IOException
```

Reads a boolean value from the input stream.

Returns:

the boolean value read.

Throws

EOFException [\(I-§2.24\)](#)

If this stream reaches the end before reading all the bytes.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readByte

```
public abstract byte readByte()  
throws IOException
```

Reads a signed 8-bit value from the input stream.

Returns:

the 8-bit value read.

Throws

EOFException [\(I-§2.24\)](#)

If this stream reaches the end before reading all the bytes.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readChar

public abstract char readChar()
throws IOException

Reads a Unicode char value from the input stream.

Returns:

the Unicode char read.

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readDouble

```
public abstract double readDouble()  
throws IOException
```

Reads a double value from the input stream.

Returns:

the double value read.

Throws

EOFException (I-§2.24)

If this stream reaches the end before reading all the bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readFloat

public abstract float readFloat()
throws IOException

Reads a float value from the input stream, high byte.

Returns:

the float value read.

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readFully

public abstract void readFully(byte b[])
throws IOException

Reads b.length bytes into the byte array. This method blocks until all the bytes are read.

Parameters:

b- the buffer into which the data is read

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public abstract void readFully(byte b[], int off, int len)
throws IOException

Reads b.length bytes into the byte array. This method blocks until all the bytes are read.

Parameters:

b- the buffer into which the data is read

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readInt

```
public abstract int readInt()  
throws IOException
```

Reads an int value from the input stream.

Returns:

the int value read.

Throws

EOFException (I-§2.24)

If this stream reaches the end before reading all the bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readLine

public abstract String readLine()
throws IOException

Reads the next line of text from the input stream.

Returns:

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readLong

public abstract long readLong()
throws IOException

Reads a long value from the input stream.

Returns:

the long value read.

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before reading all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readShort

public abstract short readShort()
throws IOException

Reads a 16-bit value from the input stream.

Returns:

the 16-bit value read.

Throws

EOFException (I-§2.24)

If this stream reaches the end before reading all the bytes.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.readUnsignedByte

```
public abstract int readUnsignedByte()  
throws IOException
```

Reads an unsigned 8-bit value from the input stream.

Returns:

the unsigned 8-bit value read.

Throws

EOFException [\(I-§2.24\)](#)

If this stream reaches the end before reading all the bytes.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readUnsignedShort

```
public abstract int readUnsignedShort()  
throws IOException
```

Reads an unsigned 16-bit value from the input stream.

Returns:

the unsigned 16-bit value read.

Throws

EOFException [\(I-§2.24\)](#)

If this stream reaches the end before reading all the bytes.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataInput.readUTF

public abstract String readUTF()
throws IOException

Reads in a string that has been encoded using a modified UTF-8 format.

Returns:

a Unicode string.

Throws

EOFException (I-§2.24)

If this stream reaches the end before reading all the bytes.

Throws

UTFDataFormatException (I-§2.31)

If the bytes do not represent a valid UTF-8 encoding of a string.

Throws

IOException (I-§2.29)

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataInput.skipBytes

```
public abstract int skipBytes(int n)  
throws IOException
```

Skips exactly n bytes of input.

Parameters:

n– the number of bytes to be skipped

Returns:

the number of bytes skipped, which is always n.

Throws

EOFException ([I-§2.24](#))

If this stream reaches the end before skipping all the bytes.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.25 Interface DataOutput

```
public interface java.io.DataOutput
{
    // Methods
    public abstract void write(byte b[]); §2.25.1
    public abstract void write(byte b[], int off, int len) §2.25.2
    public abstract void write(int b); §2.25.3
    public abstract void writeBoolean(boolean v); §2.25.4
    public abstract void writeByte(int v); §2.25.5
    public abstract void writeBytes(String s); §2.25.6
    public abstract void writeChar(int v); §2.25.7
    public abstract void writeChars(String s); §2.25.8
    public abstract void writeDouble(double v); §2.25.9
    public abstract void writeFloat(float v); §2.25.10
    public abstract void writeInt(int v); §2.25.11
    public abstract void writeLong(long v); §2.25.12
    public abstract void writeShort(int v); §2.25.13
    public abstract void writeUTF(String str); §2.25.14
}
```

The data output interface is implemented by streams that can write primitive Java data types to an output stream in a machine-independent manner.

See Also:

DataOutputStream [\(I-§2.6\)](#)

DataInput [\(I-§2.24\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutput.write

public abstract void write(byte b[])
throws IOException

Writes b.length bytes from the specified byte array to this output stream.

Parameters:

b- the data

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public abstract void write(byte b[], int off, int len)
throws IOException

Writes len bytes from the specified byte array starting at offset off to this output stream.

Parameters:

b- the data

off- the start offset in the data

len- the number of bytes to write

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

public abstract void write(int b)
throws IOException

Writes the specified byte to this data output stream.

Parameters:

b- the byte to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeBoolean

```
public abstract void writeBoolean(boolean v)  
throws IOException
```

Writes a boolean value to this output stream.

Parameters:

v – the boolean to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeByte

```
public abstract void writeByte(int v)  
throws IOException
```

Writes an 8-bit value to this output stream.

Parameters:

v – the byte value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutput.writeBytes

public abstract void writeBytes(String s)
throws IOException

Writes out the string to this output stream as a sequence of bytes.

Parameters:

s – the string of bytes to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutput.writeChar

```
public abstract void writeChar(int v)  
throws IOException
```

Writes a char value to this output stream.

Parameters:

v – the char value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeChars

public abstract void writeChars(String s)
throws IOException

Writes a string to this output stream as a sequence of characters.

Parameters:

s – the string value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutput.writeDouble

```
public abstract void writeDouble(double v)  
throws IOException
```

Writes a double value to this output stream.

Parameters:

v – the double value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DataOutput.writeFloat

```
public abstract void writeFloat(float v)  
throws IOException
```

Writes a float value to this output stream.

Parameters:

v – the float value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeInt

```
public abstract void writeInt(int v)  
throws IOException
```

Writes an int value to this output stream.

Parameters:

v – the int value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeLong

```
public abstract void writeLong(long v)  
throws IOException
```

Writes a long value to this output stream.

Parameters:

v – the long value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeShort

```
public abstract void writeShort(int v)  
throws IOException
```

Writes a 16-bit value to this output stream.

Parameters:

v – the short value to be written

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DataOutput.writeUTF

```
public abstract void writeUTF(String str)  
throws IOException
```

Writes out a Unicode string that by encoded it using modified UTF-8 format.

Parameters:

`str`- the string value to be written

Throws

IOException (I-§2.29)

If an I/O error occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.26 Interface FilenameFilter

```
public interface java.io.FilenameFilter
{
    // Methods
    public abstract boolean accept(File dir, String name); §2.26.1
}
```

Instances of classes that implement this interface are used to filter filenames. These instances are used to filter directory listings in the list method (I-§2.7.24) of class File (I-§2.7), and by the Abstract Window Toolkit's file dialog component (II-§1.15.13).

See Also:

File .

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilenameFilter.accept

public abstract boolean accept(File dir, String name)

Determines whether a specified file should be included in a file list.

Parameters:

dir- the directory in which the file was found

name- the name of the file

Returns:

true if name should be included in file list; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§2.27 Class EOFException

```
public class java.io.EOFException
    extends java.io.IOException (l-§2.29)
{
    // Constructors
    public EOFException(); §2.27.1
    public EOFException(String s); §2.27.2
}
```

Signals that an end-of-file has been reached unexpectedly during input.

This exception is mainly used by data input streams, which generally expect a binary file in a specific format, and for which an end-of-stream is an unusual condition. Most other input streams return a special value on end of stream.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


EOFException.EOFException

public EOFException()

Constructs an EOFException with no detail message.

public EOFException(String s)

Constructs an EOFException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.28 Class FileNotFoundException

```
public class java.io.FileNotFoundException
    extends java.io.IOException (l-§2.29)
{
    // Constructors
    public FileNotFoundException(); §2.28.1
    public FileNotFoundException(String s); §2.28.2
}
```

Signals that a file could not be found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileNotFoundException.FileNotFoundException

public **FileNotFoundException**()

Constructs a FileNotFoundException with no detail message.

public **FileNotFoundException**(String s)

Constructs a FileNotFoundException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.29 Class IOException

```
public class java.io.IOException  
    extends java.lang.Exception (I-§1.30)  
{  
    // Constructors  
    public IOException(); §2.29.1  
    public IOException(String s); §2.29.2  
}
```

Signals that an I/O exception of some sort has occurred.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IOException.IOException

public IOException()

Constructs an IOException with no detail message.

public IOException(String s)

Constructs an IOException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.30 Class InterruptedException

```
public class java.io.InterruptedException
    extends java.io.IOException (I-§2.29)
{
    // Fields
    public int bytesTransferred; §2.30.1

    // Constructors
    public InterruptedException(); §2.30.2
    public InterruptedException(String s); §2.30.3
}
```

Signals that an I/O operation has been interrupted.

See Also:

InputStream (I-§2.13)

OutputStream (I-§2.15)

Interrupt in class Thread (I-§1.19.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InterruptedException.bytesTransferred

public **int** bytesTransferred

Reports how many bytes had been transferred as part of the I/O operation before it was interrupted.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InterruptedException. InterruptedException

public InterruptedException()

Constructs an InterruptedException with no detail message.

public InterruptedException(String s)

Constructs an InterruptedException with the specified detail message.

Parameters:

s – the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.31 Class UTFDataFormatException

```
public class java.io.UTFDataFormatException
    extends java.io.IOException (I-§2.29)
{
    // Constructors
    public UTFDataFormatException(); §2.31.1
    public UTFDataFormatException(String s); §2.31.2
}
```

Signals that a malformed UTF-8 string has been read in a data input stream (I-§2.24) or by any class that implements the data input interface (I-§2.24). See the writeUTF method (I-§2.5.18) for the format in which UTF-8 strings are read and written.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

UTFDataFormatException.UTFDataFormatException

public UTFDataFormatException()

Constructs an UTFDataFormatException with no detail message.

public UTFDataFormatException(String s)

Constructs an UTFDataFormatException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.util

Classes

- 3.1 Class BitSet
- 3.2 Class Date
- 3.3 Class Dictionary
- 3.4 Class Hashtable
- 3.5 Class Observable
- 3.6 Class Properties
- 3.7 Class Random
- 3.8 Class Stack
- 3.9 Class StringTokenizer
- 3.10 Class Vector

Interfaces

- 3.11 Interface Enumeration
- 3.12 Interface Observer

Exceptions

- 3.13 Class EmptyStackException
- 3.14 Class NoSuchElementException

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.1 Class BitSet

```
public final class java.util.BitSet
    extends java.lang.Object (l-§1.12)
    implements java.lang.Cloneable (l-§1.22)
{
    // Constructors
    public BitSet(); §3.1.1
    public BitSet(int nbits); §3.1.2

    // Methods
    public void and(BitSet set); §3.1.3
    public void clear(int bit); §3.1.4
    public Object clone(); §3.1.5
    public boolean equals(Object obj); §3.1.6
    public boolean get(int bit); §3.1.7
    public int hashCode(); §3.1.8
    public void or(BitSet set); §3.1.9
    public void set(int bit); §3.1.10
    public int size(); §3.1.11
    public String toString(); §3.1.12
    public void xor(BitSet set); §3.1.13
}
```

This class implements a vector of bits that grows as needed. Each component of the bit set has a boolean value. The bits of a `BitSet` are indexed by nonnegative integers. Individual bits can be examined, set, or cleared.

By default, all bits in the set initially have the value false.

Every bit set has a current size, which is the number of bits currently in the bit set.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BitSet.BitSet

public `BitSet()`

Creates a new bit set. All bits are initially false.

public `BitSet(int nbits)`

Creates a bit set whose initial size is the specified number of bits. All bits are initially false.

Parameters:

`nbits`- the initial size of the bit set.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.and

public void and(BitSet set)

Performs a logical **and** of this target bit set with the argument bit set. This bit set is modified so that each bit in it has the value true if it both initially had the value true and the corresponding bit in the bit set argument also had the value true.

Parameters:

set- a bit set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.clear

public void clear(int bit)

Sets the bit specified by the index to false.

Parameters:

`bit`– the index of the bit to be cleared

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.clone

public Object clone()

The clone of the bit set is another bit set that has exactly the same bits set to true as this bit set and the same current size (I-§3.1.11).

Returns:

a clone of this bit set.

Overrides:

clone in class Object (I-§1.12.2).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

BitSet.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a BitSet object that has exactly the same set of bits set to true as this bit set. The current sizes (I-§3.1.11) of the two bit sets are not compared.

Parameters:

obj- the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.get

public boolean get(int bit)

Parameters:

bit- the bit index

Returns:

the value of the bit with the specified index.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.hashCode

public int hashCode()

Returns:

a hash code value for this bit set.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.or

public void or(BitSet set)

Performs a logical **or** of this bit set with the bit set argument. This bit set is modified so that a bit in it has the value true if it either already had the value true or the corresponding bit in the bit set argument has the value true.

Parameters:

set- a bit set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.set

public void set(int bit)

Sets the bit specified by the index to true.

Parameters:

bit- a bit index

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.size

public int size()

Returns:

the number of bits currently in this bit set.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.toString

public String toString()

Returns:

a string representation of this bit set.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BitSet.xor

public void xor(BitSet set)

Performs a logical **xor** of this bit set with the bit set argument. This bit set is modified so that a bit in it has the value true if one of the following statements holds:

- The bit initially has the value true, and the corresponding bit in the argument has the value false.
- The bit initially has the value false, and the corresponding bit in the argument has the value true.

Parameters:

set- a bit set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.2 Class Date

```
public class java.util.Date
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public Date(); §3.2.1
    public Date(int year, int month, int date); §3.2.2
    public Date(int year, int month, int date, §3.2.3
                int hrs, int min);
    public Date(int year, int month, int date, §3.2.4
                int hrs, int min, int sec);
    public Date(long date); §3.2.5
    public Date(String s); §3.2.6

    // Methods
    public boolean after(Date when); §3.2.7
    public boolean before(Date when); §3.2.8
    public boolean equals(Object obj); §3.2.9
    public int getDate(); §3.2.10
    public int getDay(); §3.2.11
    public int getHours(); §3.2.12
    public int getMinutes(); §3.2.13
    public int getMonth(); §3.2.14
    public int getSeconds(); §3.2.15
    public long getTime(); §3.2.16
    public int getTimezoneOffset(); §3.2.17
    public int getYear(); §3.2.18
    public int hashCode(); §3.2.19
    public static long parse(String s); §3.2.20
    public void setDate(int date); §3.2.21
    public void setHours(int hours); §3.2.22
    public void setMinutes(int minutes); §3.2.23
    public void setMonth(int month); §3.2.24
    public void setSeconds(int seconds); §3.2.25
    public void setTime(long time); §3.2.26
    public void setYear(int year); §3.2.27
    public String toGMTString(); §3.2.28
    public String toLocaleString(); §3.2.29
    public String toString(); §3.2.30
    public static long UTC(int year, int month, int date, §3.2.31
                        int hrs, int min, int sec);
}
```

The class Date provides an abstraction of dates and times. Dates may be constructed from a year, month, date (day of month), hour, minute, and second. Those six components, as well as the day of

the week, may be extracted from a date. Dates may also be compared and converted to a readable string form. A date is represented to a precision of one millisecond.

To print today's date:

```
System.out.println("today = " + new Date());
```

To find out the day of the week for some particular date, for example, January 16, 1963:

```
new Date(63, 0, 16).getDay()
```

While the `Date` class is intended to reflect UTC (Coordinated Universal Time), it may not do so exactly, depending on the host environment of the Java Virtual Machine. Nearly all modern operating systems assume that $1 \text{ day} = 24 \times 60 \times 60 = 86400$ seconds in all cases. In UTC, however, about once every year or two there is an extra second, called a "leap second." The leap second is always added as the last second of the day, and always on December 31 or June 30. For example, the last minute of the year 1995 was 61 seconds long, thanks to an added leap second. Most computer clocks are not accurate enough to be able to reflect the leap-second distinction.

Some computer standards are defined in terms of GMT (Greenwich Mean Time), which is equivalent to UT (Universal Time). GMT is the "civil" name for the standard; UT is the "scientific" name for the same standard. The distinction between UTC and UT is that UTC is based on an atomic clock and UT is based on astronomical observations, which for all practical purposes is an invisibly fine hair to split. Because the earth's rotation is not uniform-it slows down and speeds up in complicated ways; UT does not always flow uniformly. Leap seconds are introduced as needed into UTC so as to keep UTC within 0.9 seconds of UT1, which is a version of UT with certain corrections applied. There are other time and date systems as well; for example, the time scale used by GPS (the satellite-based Global Positioning System) is synchronized to UTC but is not adjusted for leap seconds. An interesting source of further information is the US Naval Observatory, particularly the Directorate of Time at

<http://tycho.usno.navy.mil>

and their definitions of "Systems of Time" at

<http://tycho.usno.navy.mil/systime.html>

In all methods of class `Date` that accept or return year, month, date, hours, minutes, and seconds values, the following representations are used:

- A year y is represented by the integer $y - 1900$.
- A month is represented by an integer from 0 to 11; 0 is January, 1 is February, and so on; thus 11 is December.
- A date (day of month) is represented by an integer from 1 to 31 in the usual manner.
- An hour is represented by an integer from 0 to 23. Thus the hour from midnight to 1 AM is hour 0, and the hour from noon to 1 PM is hour 12.
- A minute is represented by an integer from 0 to 59 in the usual manner.
- A second is represented by an integer from 0 to 60; the value 60 occurs only for leap seconds and even then only in Java implementations that actually track leap seconds correctly.

In all cases, arguments given to methods for these purposes need not fall within the indicated

ranges; for example, a date may be specified as January 32 and is interpreted as meaning February 1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Date.Date

public Date()

Allocates a Date object and initializes it so that it represents the time at which it was allocated measured to the nearest millisecond.

See Also:

currentTimeMillis ([I-§1.18.5](#)) in class System.

public Date(int year, int month, int date)

Allocates a Date object and initializes it so that it represents midnight, local time, at the beginning of the day specified by the year, month, and date arguments.

Parameters:

year- the year minus 1900

month- a month between 0-11

date- day of the month between 1-31

public Date(int year, int month, int date, int hrs, int min)

Allocates a Date object and initializes it so that it represents the specified hour and minute, local time, of the date specified by the year, month, and date arguments.

Parameters:

year- the year minus 1900

month- a month between 0-11

date- day of the month between 1-31

hrs- hours between 0-23

min- minutes between 0-59

public Date(int year, int month, int date, int hrs, int min, int sec)

Allocates a Date object and initializes it so that it represents the specified hour, minute, and second, local time of the date specified by the year, month, and date arguments.

Parameters:

year- the year minus 1900
month- a month between 0-11
date- day of the month between 1-31
hrs- hours between 0-23
min- minutes between 0-59
sec- seconds between 0-59

public Date(long date)

Allocates a Date object and initializes it to represent the specified number of milliseconds since January 1, 1970, 00:00:00GMT.

Parameters:

date- milliseconds since January 1, 1970, 00:00:00 GMT

See Also:

currentTimeMillis ([I-§1.18.5](#)) in class System.

public Date(String s)

Allocates a Date object and initializes it so that it represents the date and time indicated by the string s, which is interpreted as if by the parse method ([I-§3.2.20](#)).

Parameters:

s- a string representation of the date

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.after

public boolean after(Date when)

Parameters:

when- a date

Returns:

true if this date is after the argument date; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.before

public boolean before (Date when)

Parameters:

when- a date

Returns:

true if this date is before the argument date; false otherwise.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Date.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a Date object that represents the same point in time, to the millisecond, as this object.

Thus two Date objects are equal if the getTime method (I-§3.2.16) returns the same long value for both.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getDate

```
public int getDate()
```

Returns:

the day of the month represented by this date. The value returned is between 1 and 31.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Date.getDay

public int **getDay**()

Returns:

the day of the week represented by this date. The value returned is between 0 and 6, where 0 represents Sunday.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getHours

public int `getHours()`

Returns:

the hour represented by this date. The value returned is between 0 and 23, where 0 represents midnight.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getMinutes

public int **getMinutes**()

Returns:

the number of minutes past the hour represented by this date. The value returned is between 0 and 59.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getMonth

```
public int getMonth()
```

Returns:

the month represented by this date. The value returned is between 0 and 11, with the value 0 representing January.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Date.getSeconds

public int getSeconds()

Returns:

the number of seconds past the minute represented by this date. The value returned is between 0 and 60. The value 60 can only occur on those Java Virtual Machines that take leap seconds into account.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getTime

public long getTime()

Returns:

the number of milliseconds since January 1, 1970, 00:00:00 GMT, represented by this date.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getTimezoneOffset

public int getTimezoneOffset()

Determines the local time zone offset. The time zone offset is the number of minutes that must be added to Greenwich Mean Time to give the local time zone. This value includes the correction, if necessary, for daylight savings time.

Returns:

the time zone offset, in minutes, for the current locale.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.getYear

```
public int getYear()
```

Returns:

the year represented by this date, minus 1900.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Date.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.parse

public static long parse(String s)

Given a string representing a time, parses it and returns the time value. This method recognizes most standard syntypes.

It accepts many syntypes; in particular, it recognizes the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT." It also understands the continental US time zone abbreviations, but for general use, a time zone offset should be used: "Sat, 12 Aug 1995 13:30:00 GMT+0430" (4 hours, 30 minutes west of the Greenwich meridian). If no time zone is specified, the local time zone is assumed. GMT and UTC are considered equivalent.

Parameters:

s - a string to be parsed as a date

Returns:

the number of milliseconds since January 1, 1970, 00:00:00 GMT, represented by the string argument.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setDate

public void setDate(int date)

Sets the day of the month of this date to the specified value.

Parameters:

date- the day value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setHours

public void setHours(int hours)

Sets the hour of this date to the specified value.

Parameters:

hours- the hour value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setMinutes

public void setMinutes(int minutes)

Sets the minutes of this date to the specified value.

Parameters:

minutes- the value of the minutes

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setMonth

public void setMonth(int month)

Sets the month of this date to the specified value.

Parameters:

`month`– the month value (0-11)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setSeconds

public void setSeconds(int seconds)

Sets the seconds of this date to the specified value.

Parameters:

seconds- the second value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setTime

public void setTime(long time)

Sets this date to represent the specified number of milliseconds since January 1, 1970 00:00:00 GMT.

Parameters:

time- A number of milliseconds

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.setYear

public void setYear(int year)

Sets the year of this date to be the specified value plus 1900.

Parameters:

year- the year value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.toGMTString

public String toGMTString()

Creates a string representation of this date. The result is of the form:

"12 Aug 1995 02:30:00 GMT"
the day of the month is always one or two digits. The other fields have exactly the width shown.
The time zone is always given as "GMT."

Returns:

A string representation of this date, using the Internet GMT conventions.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.toLocaleString

public String toLocaleString()

Creates a string representation of this date in an implementation-dependent form. The intent is that the form should be familiar to the user of the Java application, wherever it may happen to be running. The intent is comparable to that of the %c format supported by the strftime() function of ISO C.

Returns:

A string representation of this date, using the locale conventions.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.toString

public String toString()

Creates a canonical string representation of the date. The result is of the form "Sat Aug 12 02:30:00 PDT 1995."

Returns:

A string representation of this date.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Date.UTC

public static long

UTC(int year, int month, int date, int hrs, int min, int sec)

Determines the date and time based on the arguments. The arguments are interpreted in UTC, not in the local time zone.

Parameters:

year- the year minus 1900

month- a month between 0-11

date- day of the month between 1-31

hrs- hours between 0-23

min- minutes between 0-59

sec- seconds between 0-59

Returns:

the number of seconds since January 1, 1970, 00:00:00 GMT, for the date and time specified by the arguments.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.3 Class Dictionary

```
public abstract class java.util.Dictionary
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public Dictionary(); §3.3.1

    // Methods
    public abstract Enumeration elements(); §3.3.2
    public abstract Object get(Object key); §3.3.3
    public abstract boolean isEmpty(); §3.3.4
    public abstract Enumeration keys(); §3.3.5
    public abstract Object put(Object key, Object value) §3.3.6
    public abstract Object remove(Object key); §3.3.7
    public abstract int size(); §3.3.8
}
```

The Dictionary class is the abstract parent of any class, such as Hashtable (I-§3.4), which maps keys to values. Any non-null object can be used as a key and as a value.

As a rule, the equals method (I-§1.12.3) should be used by implements of this class to decide if two keys are the same.

See Also:

hashCode in class Object (I-§1.12.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dictionary.Dictionary

public Dictionary()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dictionary.elements

public abstract Enumeration elements()

Returns:

An enumeration ([I-§3.11](#)) of the values in the dictionary.

See Also:

keys ([I-§3.3.5](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dictionary.get

public abstract Object get(Object key)

Parameters:

key- a key in this dictionary

Returns:

the value to which the key is mapped in this dictionary; null if the key is not mapped to any value in this dictionary.

See Also:

put ([I-§3.3.6](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dictionary.isEmpty

public abstract boolean isEmpty()

Returns:

true if this dictionary maps no keys to values; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dictionary.keys

public abstract Enumeration keys()

Returns:

An enumeration ([I-§3.11](#)) of the keys in this dictionary.

See Also:

elements ([I-§3.3.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dictionary.put

public abstract Object put(Object key, Object value)

Maps the specified key to the specified value in this dictionary. Neither the key nor the value can be null.

The value can be retrieved by calling the the get [method \(I-§3.3.3\)](#) with a key that is equal [\(I-§1.12.3\)](#) to the original key.

Parameters:

key- the hash table key

value- the value

Returns:

the previous value to which the key was mapped in the dictionary, or null if the key did not have a previous mapping.

Throws

NullPointerException [\(I-§1.40\)](#)

If the key or value is null.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dictionary.remove

public abstract Object remove(Object key)

Removes the key (and its corresponding value) from this dictionary. This method does nothing if the key is not in this dictionary.

Parameters:

key– the key that needs to be removed

Returns:

the value to which the key had been mapped in this dictionary, or null if the key did not have a mapping.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dictionary.size

public abstract int size()

Returns:

the number of keys in this dictionary.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.4 Class Hashtable

```
public class java.util.Hashtable
    extends java.util.Dictionary (I-§3.3)
    implements java.lang.Cloneable (I-§1.22)
{
    // Constructors
    public Hashtable(); §3.4.1
    public Hashtable(int initialCapacity); §3.4.2
    public Hashtable(int initialCapacity, float loadFactor); §3.4.3

    // Methods
    public void clear(); §3.4.4
    public Object clone(); §3.4.5
    public boolean contains(Object value); §3.4.6
    public boolean containsKey(Object key); §3.4.7
    public Enumeration elements(); §3.4.8
    public Object get(Object key); §3.4.9
    public boolean isEmpty(); §3.4.10
    public Enumeration keys(); §3.4.11
    public Object put(Object key, Object value); §3.4.12
    protected void rehash(); §3.4.13
    public Object remove(Object key); §3.4.14
    public int size(); §3.4.15
    public String toString(); §3.4.16
}
```

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.

To successfully store and retrieve objects from a hash table, the objects used as keys must implement the hashCode method (I-§1.12.6) and the equals method (I-§1.12.3).

An instance of Hashtable has two parameters that affect its efficiency: its *capacity* and its *load factor*. The load factor should be between 0.0 and 1.0. When the number of entries in the hash table exceeds the product of the load factor and the current capacity, the capacity is increased by calling the rehash method (I-§3.4.13). Larger load factors use memory more efficiently, at the expense of larger expected time per lookup.

If many entries are to be made into a hash table, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

This example creates a hash table of numbers. It uses the names of the numbers as keys:

```
Hashtable numbers = new Hashtable();

numbers.put("one", new Integer(1));

numbers.put("two", new Integer(2));
```



```
numbers.put("three", new Integer(3));
```

To retrieve a number use the following code:

```
Integer n = (Integer)numbers.get("two");  
if (n != null) {  
    System.out.println("two = " + n);  
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Hashtable.Hashtable

public Hashtable()

Constructs a new empty hash table.

public Hashtable(int initialCapacity)

Constructs a new, empty hash table with the specified initial capacity.

Parameters:

initialCapacity- the initial capacity of the hash table

public

Hashtable(int initialCapacity, float loadFactor)

Constructs a new, empty hash table with the specified initial capacity and the specified load factor.

Parameters:

initialCapacity- the initial size of the hash table

loadFactor- a number between 0.0 and 1.0

Throws

IllegalArgumentException (I-§1.32)

If the initial capacity is less than or equal to zero, or if the load factor is less than or equal to zero.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.clear

public void clear()

Clears this hash table so that it contains no keys.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Hashtable.clone

public Object clone()

Creates a shallow copy of this hash table. The keys and values themselves are not cloned.

Returns:

a clone of the hash table.

Overrides:

clone in class Object ([I-§1.12.2](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Hashtable.contains

public boolean contains(Object value)

Parameters:

value- a value to search for

Returns:

true if some key maps to the value argument in this hash table; false otherwise.

Throws

NullPointerException ([I-§1.40](#))

If the value is null.

See Also:

containsKey ([I-§3.4.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.ContainsKey

public boolean **containsKey**(Object **key**)

Parameters:

key– possible key

Returns:

true if the specified object is a key in this hash table; false otherwise.

See Also:

contains (I-§3.4.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.elements

public `Enumeration elements()`

Returns:

an enumeration ([I-§3.11](#)) of the values in this hash table.

Overrides:

elements in class Dictionary ([I-§3.3.2](#)).

See Also:

keys ([I-§3.4.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.get

public Object get(Object key)

Parameters:

key- a key in the hash table

Returns:

the value to which the key is mapped in this hash table; null if the key is not mapped to any value in this hash table.

Overrides:

get in class Dictionary ([I-§3.3.3](#)).

See Also:

put ([I-§3.4.12](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.IsEmpty

public boolean isEmpty()

Returns:

true if this hash table maps no keys to values; false otherwise.

Overrides:

isEmpty in class Dictionary ([I-§3.3.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.keys

public **Enumeration** **keys()**

Returns:

an enumeration ([I-§3.11](#)) of the keys in this hash table.

Overrides:

keys in class Dictionary ([I-§3.3.5](#)).

See Also:

elements ([I-§3.4.8](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.put

public Object put(Object key, Object value)

Maps the specified key to the specified value in this hash table. Neither the key nor the value can be null.

The value can be retrieved by calling the get [method \(I-§3.4.9\)](#) with a key that is equal [\(I-§1.12.3\)](#) to the original key.

Parameters:

key- the hash table key

value- the value

Returns:

the previous value of the specified key in this hash table, or null if it did not have one.

Throws

NullPointerException [\(I-§1.40\)](#)

If the key or value is null.

Overrides:

put in class Dictionary [\(I-§3.3.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.rehash

protected void rehash()

Rehashes the contents of the hash table into a hash table with a larger capacity. This method is called automatically when the number of keys in the hash table exceeds this hash table's capacity and load factor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.remove

public Object remove(Object key)

Removes the key (and its corresponding value) from this hash table. This method does nothing if the key is not in the hash table.

Parameters:

key- the key that needs to be removed

Returns:

the value to which the key had been mapped in this hash table, or null if the key did not have a mapping.

Overrides:

remove in class Dictionary ([I-§3.3.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.size

public int size()

Returns:

the number of keys in this hash table.

Overrides:

size in class Dictionary (I-§3.3.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Hashtable.toString

public String toString()

Returns:

a string representation of this hash table.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.5 Class Observable

```
public class java.util.Observable
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public Observable(); §3.5.1

    // Methods
    public void addObserver(Observer o); §3.5.2
    protected void clearChanged(); §3.5.3
    public int countObservers(); §3.5.4
    public void deleteObserver(Observer o); §3.5.5
    public void deleteObservers(); §3.5.6
    public boolean hasChanged(); §3.5.7
    public void notifyObservers(); §3.5.8
    public void notifyObservers(Object arg); §3.5.9
    protected void setChanged(); §3.5.10
}
```

This class represents an observable object, or "data" in the model-view paradigm. It can be subclassed to represent an object that the application wants to have observed.

An observable object can have one or more observers (I-§3.12). After an observable instance changes, an application calling the Observable's notifyObservers method (§3.5.8, §3.5.9) causes all of its observers to be notified of the change by a call to their update method (I-§3.12.1).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Observable.Observable

public `Observable()`

The default constructor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Observable.addObserver

public void addObserver (Observer o)

Adds an observer to the set of observers for this object.

Parameters:

- – an observer to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.clearChanged

protected void clearChanged()

Indicates that this object has no longer changed, or that it has already notified all of its observers of its most recent change. This method is called automatically by the notifyObservers methods (§3.5.8, §3.5.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.countObservers

public int countObservers()

Returns:

the number of observers of this object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.deleteObserver

public void deleteObserver (Observer o)

Deletes an observer from the set of observers of this object.

Parameters:

- – the observer to be deleted

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.deleteObservers

public void deleteObservers ()

Clears the observer list so that this object no longer has any observers.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.hasChanged

public **boolean** hasChanged()

Determines if this object has changed.

Returns:

true if the setChanged method (I-§3.5.10) has been called more recently than the clearChanged (I-§3.5.3) method on this object; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.notifyObservers

public void notifyObservers ()

If this object has changed, as indicated by the hasChanged method (I-§3.5.7), then notify all of its observers and then call the clearChanged method (I-§3.5.3) to indicate that this object has no longer changed.

Each observer has its update method (I-§3.12.1) called with two arguments: the observable object and null.

public void notifyObservers (Object arg)

If this object has changed, as indicated by the hasChanged method (I-§3.5.7), then notify all of its observers and then call the clearChanged method (I-§3.5.3) to indicate that this object has no longer changed.

Each observer has its update method (I-§3.12.1) called with two arguments: the observable object and the arg argument.

Parameters:

arg- any object

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Observable.setChanged

protected void setChanged()

Indicates that this object has changed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.6 Class Properties

```
public class java.util.Properties
    extends java.util.Hashtable (l-§3.4)
{
    // Fields
    protected Properties defaults; §3.6.1

    // Constructors
    public Properties(); §3.6.2
    public Properties(Properties defaults); §3.6.3

    // Methods
    public String getProperty(String key); §3.6.4
    public String getProperty(String key, String defaultValue); §3.6.5
    public void list(PrintStream out); §3.6.6
    public void load(InputStream in); §3.6.7
    public Enumeration propertyNames(); §3.6.8
    public void save(OutputStream out, String header); §3.6.9
}
```

The Properties class represents a persistent set of properties. The Properties can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string.

A property list can contain another property list as its "defaults"; this second property list is searched if the property key is not found in the original property list.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Properties.defaults

protected Properties defaults

A property list which contains default values for any keys not found in this property list.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Properties.Properties

public Properties()

Creates an empty property list with no default values.

public Properties(Properties defaults)

Creates an empty property list with the specified defaults.

Parameters:

defaults- the defaults

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Properties.GetProperty

public String GetProperty(String key)

Searches for the property with the specified key in this property list.

Parameters:

key- the property key

Returns:

the value in this property list with the specified key value. If the key is not found in this property list, the default property list ([I-§3.6.1](#)), and its defaults, recursively, are then checked. The method returns null if the property is not found.

public String GetProperty(String key, String defaultValue)

Searches for the property with the specified key in this property list.

Parameters:

key- the hash table key

defaultValue- a default value

Returns:

the value in this property list with the specified key value. If the key is not found in this property list, the default property list ([I-§3.6.1](#)), and its defaults, recursively, are then checked. The method returns the default value argument if the property is not found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Properties.list

public void list(PrintStream out)

Prints this property list out to the specified output stream. This method is useful for debugging.

Parameters:

out- an output stream

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Properties.load

public void load(InputStream in)
throws IOException

Reads a property list from an input stream.

Parameters:

in- the input stream

Throws

IOException ([I-§2.29](#))

If an error occurred when reading from the input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Properties.propertyNames

public Enumeration propertyNames()

Returns:

An enumeration ([I-§3.11](#)) of all the keys in this property list, including the keys in the default property list ([I-§3.6.1](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Properties.save

public void save(OutputStream out, String header)

Stores this property list to the specified output stream. The string header is printed as a comment at the beginning of the stream.

Parameters:

out- an output stream

header- a description of the property list

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.7 Class Random

```
public class java.util.Random
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public Random(); §3.7.1
    public Random(long seed); §3.7.2

    // Methods
    public double nextDouble(); §3.7.3
    public float nextFloat(); §3.7.4
    public double nextGaussian(); §3.7.5
    public int nextInt(); §3.7.6
    public long nextLong(); §3.7.7
    public void setSeed(long seed); §3.7.8
}
```

An instance of this class is used to generate a stream of pseudo-random numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.

If two instances of Random are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers.

Many applications will find the random method (I-§1.10.26) in class Math simpler to use.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Random.Random

public Random()

Creates a new random number generator. Its seed is initialized to a value based on the current time ([I-§1.18.5](#)).

public Random(long seed)

Creates a new random number generator using a single long seed.

Parameters:

seed- the initial seed

See Also:

setSeed ([I-§3.7.8](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.nextDouble

public double nextDouble()

Returns:

the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.nextFloat

public float nextFloat()

Returns:

the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.nextGaussian

public double nextGaussian()

Returns:

the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.nextInt

public int nextInt()

Returns:

the next pseudorandom, uniformly distributed int value from this random number generator's sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.nextLong

public long nextLong()

Returns:

the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Random.setSeed

public void setSeed(long seed)

Sets the seed of this random number generator using a single long seed.

Parameters:

seed– the initial seed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.8 Class Stack

```
public class java.util.Stack
    extends java.util.Vector (l-§3.10)
{
    // Constructors
    public stack(); §3.8.1

    // Methods
    public boolean empty(); §3.8.2
    public Object peek(); §3.8.3
    public Object pop(); §3.8.4
    public Object push(Object item); §3.8.5
    public int search(Object o); §3.8.6
}
```

The Stack class represents a last-in-first-out (LIFO) stack of objects.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Stack.Stack

public `stack()`

Creates a new stack with no elements.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Stack.empty

public boolean empty()

Returns:

true if this stack is empty; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Stack.peek

public Object peek ()

Looks at the object at the top of this stack without removing it from the stack.

Returns:

the object at the top of this stack.

Throws

EmptyStackException (I-§3.13)

If this stack is empty.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Stack.pop

public Object pop()

Removes the object at the top of this stack and returns that object as the value of this function.

Returns:

The object at the top of this stack.

Throws

EmptyStackException (I-§3.13)

If this stack is empty.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Stack.push

public Object push(Object item)

Pushes an item onto the top of this stack.

Parameters:

item- the item to be pushed onto this stack.

Returns:

the item argument.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Stack.search

public int search(Object o)

Determines if an object is on this stack.

Parameters:

- the desired object

Returns:

The distance from the top of the stack at which the object is located; the return value -1 indicates that the object is not on the stack.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.9 Class StringTokenizer

```
public class java.util.StringTokenizer
    extends java.lang.Object (I-§1.12)
    implements java.util.Enumeration (I-§3.11)
{
    // Constructors
    public StringTokenizer(String str); §3.9.1
    public StringTokenizer(String str, String delim); §3.9.2
    public StringTokenizer(String str, String delim, §3.9.3
                           boolean returnTokens);
    // Methods
    public int countTokens(); §3.9.4
    public boolean hasMoreElements(); §3.9.5
    public boolean hasMoreTokens(); §3.9.6
    public Object nextElement(); §3.9.7
    public String nextToken(); §3.9.8
    public String nextToken(String delim); §3.9.9
}
```

The string tokenizer class allows an application to break a string into tokens. The tokenization method is much simpler than the one used by the StreamTokenizer class (I-§2.22). The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor does it recognize and skip comments.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnTokens flag having the value true or false:

- If the flag is false, delimiter characters merely serve to separate tokens. A token is a maximal sequence of consecutive characters that are delimiters.
- If the flag is true, delimiters characters are considered to be tokens. A token is either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

The following is one example of the use of the tokenizer. The code:

```
StringTokenizer st = new StringTokenizer("this is a test");
while (st.hasMoreTokens()) {
    println(st.nextToken());
}
```

Prints the following output:

```
this
```


is
a
test

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringTokenizer.StringTokenizer

public StringTokenizer(String str)

Constructs a string tokenizer for the specified string. The tokenizer uses the default delimiter set, which is " \t\n\r", the space character, the tab character, the newline character, and the carriage return character.

Parameters:

str- a string to be parsed

public StringTokenizer(String str, String delim)

Constructs a string tokenizer for the specified string. The characters in the delim argument are the delimiters for separating tokens.

Parameters:

str- a string to be parsed

delim- the delimiters

public StringTokenizer(String str, String delim, boolean returnTokens)

Constructs a string tokenizer for the specified string. The characters in the delim argument are the delimiters for separating tokens.

If the returnTokens flag is true, then the delimiter characters are also returned as tokens. Each delimiter is returned as a string of length one. If the flag is false, the delimiter characters are skipped and only serve as separators between tokens.

Parameters:

str- a string to be parsed

delim- the delimiters

returnTokens- flag indicating whether to return the delimiters as tokens


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


StringTokenizer.countTokens

public int countTokens()

Calculates the number of times that this tokenizer's nextToken method (I-§3.9.8) can be called before it generates an exception.

Returns:

the number of tokens remaining in the string using the current delimiter set.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringTokenizer.hasMoreElements

public boolean hasMoreElements ()

This method returns the same value as the following hasMoreTokens method. It exists so that this class can implement the enumeration (I-§3.11) interface.

Returns:

true if there are more tokens; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringTokenizer.hasMoreTokens

public boolean hasMoreTokens ()

Returns:

true if there are more tokens available from this tokenizer's string; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringTokenizer.nextElement

public Object nextElement()

This method returns the same value as the following nextToken method, except that its declared return value is Object rather than String. It exists so that this class can implement the enumeration (I-§3.11) interface.

Returns:

the next token in the string.

Throws

NoSuchElementException (I-§3.14)

If there are no more tokens in this tokenizer's string.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

StringTokenizer.nextToken

public String nextToken()

Returns:

the next token from this string tokenizer.

Throws

NoSuchElementException ([I-§3.14](#))

If there are no more tokens in this tokenizer's string.

public String nextToken(String delim)

Gets the next token in this string tokenizer's string. The new delimiter set remains the default after this call.

Parameters:

`delim`- the new delimiters

Returns:

the next token, after switching to the new delimiter set.

Throws

NoSuchElementException ([I-§3.14](#))

If there are no more tokens in the string.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.10 Class Vector

```
public class java.util.Vector
    extends java.lang.Object (l-§1.12)
    implements java.lang.Cloneable (l-§1.22)
{
    // Fields
    protected int capacityIncrement; §3.10.1
    protected int elementCount; §3.10.2
    protected Object elementData[]; §3.10.3

    // Constructors
    public Vector(); §3.10.4
    public Vector(int initialCapacity); §3.10.5
    public Vector(int initialCapacity, §3.10.6
                  int capacityIncrement);

    // Methods
    public final void addElement(Object obj); §3.10.7
    public final int capacity(); §3.10.8
    public Object clone(); §3.10.9
    public final boolean contains(Object elem); §3.10.10
    public final void copyInto(Object anArray[]); §3.10.11
    public final Object elementAt(int index); §3.10.12
    public final Enumeration elements(); §3.10.13
    public final void ensureCapacity(int minCapacity) §3.10.14
    public final Object firstElement(); §3.10.15
    public final int indexOf(Object elem); §3.10.16
    public final int indexOf(Object elem, int index); §3.10.17
    public final void insertElementAt(Object obj, int index); §3.10.18
    public final boolean isEmpty(); §3.10.19
    public final Object lastElement(); §3.10.20
    public final int lastIndexOf(Object elem); §3.10.21
    public final int lastIndexOf(Object elem, int index); §3.10.22
    public final void removeAllElements(); §3.10.23
    public final boolean removeElement(Object obj); §3.10.24
    public final void removeElementAt(int index); §3.10.25
    public final void setElementAt(Object obj, int index); §3.10.26
    public final void setSize(int newSize); §3.10.27
    public final int size(); §3.10.28
    public final String toString(); §3.10.29
    public final void trimToSize(); §3.10.30
}
```

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as

needed to accommodate adding and removing items after the Vector has been created.

Each vector tries to optimize storage management by maintaining a capacity and a capacityIncrement. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.capacityIncrement

protected int capacityIncrement

The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity. If the capacity is 0, the capacity of the vector is doubled each time it needs to grow.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.elementAtCount

protected int elementCount

The number of valid components in the vector.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.elementAt

protected Object elementData[]

The array buffer into which the components of the vector are stored. The length of this array buffer is the (current) capacity of the vector.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.Vector

public Vector()

Constructs an empty vector.

public Vector(int initialCapacity)

Constructs an empty vector. Its initial capacity is the specified argument size.

Parameters:

initialCapacity- the initial capacity of the vector

public Vector(int initialCapacity, int capacityIncrement)

Constructs an empty vector with the specified capacity and the specified capacity increment.

Parameters:

initialCapacity- the initial capacity of the vector

capacityIncrement- the amount by which the capacity is increased when the vector overflows

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.addElement

public final void addElement(Object obj)

Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

Parameters:

obj- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.capacity

```
public final int capacity()
```

Returns:

the current capacity of this vector.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.clone

public Object clone()

Returns:

a clone of this vector.

Overrides:

clone() in class Object ([I-§1.12.2](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Vector.contains

```
public final boolean contains(Object elem)
```

Parameters:

elem- an object

Returns:

true if the specified object is a component in this vector; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.copyInto

```
public final void copyInto(Object anArray[])
```

Copies the components of this vector into the specified array. The array must be big enough to hold all the objects in this vector.

Parameters:

`anArray`- the array into which the components get copied

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.elementAt

`public final Object elementAt(int index)`

Parameters:

`index`— an index into this vector

Returns:

the component at the specified index.

Throws

`ArrayIndexOutOfBoundsException` ([I-§1.25](#))

If an invalid index was given.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.elements

```
public final Enumeration elements()
```

Returns:

an enumeration (I-§3.11) of the components of this vector.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.ensureCapacity

public final void ensureCapacity(int minCapacity)

Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

Parameters:

minCapacity- the desired minimum capacity

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.firstElement

`public final Object firstElement()`

Returns:

the first component of this vector.

Throws

NoSuchElementException ([I-§3.14](#))

If this vector has no components.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.indexOf

```
public final int indexOf(Object elem)
```

Parameters:

elem- an object

Returns:

the index of the first occurrence of the argument in this vector; returns -1 if the object is not found.

```
public final int indexOf(Object elem, int index)
```

Parameters:

elem- an object

index- the index where to start searching

Returns:

the index of the first occurrence of the object argument in this vector at position index or later in the vector; returns -1 if the object is not found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.insertElementAt

public final void insertElementAt(Object obj, int index)

Inserts the specified object as a component in this vector at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

The index must be a value greater than or equal to 0 and less than or equal to the current size (I-§3.10.28) of the vector.

Parameters:

obj- the component to insert

index- where to insert the new component

Throws

ArrayIndexOutOfBoundsException (I-§1.25)

If the index was invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.isEmpty

```
public final boolean isEmpty()
```

Returns:

true if this vector has no components; false otherwise.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.lastElement

public final Object lastElement()

Returns:

the last component of the vector, i.e. the component at index `size() - 1`.

Throws

NoSuchElementException (I-§3.14)

If this vector is empty.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.lastIndexOf

```
public final int lastIndexOf(Object elem)
```

Parameters:

`elem` - the desired component

Returns:

the index of the last occurrence of the argument in this vector; returns -1 if the object is not found.

```
public final int lastIndexOf(Object elem, int index)
```

Searches backwards for the specified object, starting from the specified index and returns an index to it.

Parameters:

`elem` - the desired component

`index` - the index where to start searching

Returns:

the index of the last occurrence of the object argument in this vector at position less than index in the vector; returns -1 if the object is not found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.removeAllElements

public final void removeAllElements()

Removes all components from this vector and sets its size to zero.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.removeElement

public final boolean removeElement(Object obj)

Removes the first occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.

Parameters:

obj – the component to be removed

Returns:

true if the argument was a component of this vector; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.removeElementAt

public **final** **void** **removeElementAt**(**int** **index**)

Deletes the component at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted downward to have an index one smaller than the value it had previously.

The index must be a value greater than or equal to 0 and less than the current size (I-§3.10.28) of the vector.

Parameters:

index- the index of the object to remove

Throws

ArrayIndexOutOfBoundsException (I-§1.25)

If the index was invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.setElementAt

`public final void setElementAt(Object obj, int index)`

Sets the component at the specified index of this vector to be the specified object. The previous component at that position is discarded.

The index must be a value greater than or equal to 0 and less than the current size (I-§3.10.28) of the vector.

Parameters:

`obj`– what the component is to be set to

`index`– the specified index

Throws

`ArrayIndexOutOfBoundsException` (I-§1.25)

If the index was invalid.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.setSize

public final void setSize(int newSize)

Sets the size of this vector. If the new size is greater than the current size, new null items are added to the end of the vector. If the new size is less than the current size, all components at index newSize and greater are discarded.

Parameters:

newSize- the new size of this vector

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.size

```
public final int size()
```

Returns:

the number of components in this vector.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Vector.toString

public final String toString()

Creates a string representation of this vector.

Returns:

a string representation of this vector.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Vector.trimToSize

public final void trimToSize()

Trims the capacity of this vector to be the vector's current size (I-§3.10.28). An application can use this operation to minimize the storage of a vector.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.11 Interface Enumeration

```
public interface java.util.Enumeration
{
    // Methods
    public abstract boolean hasMoreElements(); §3.11.1
    public abstract Object nextElement(); §3.11.2
}
```

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the nextElement method (I-§3.11.2) return successive elements of the series.

For example, to print all elements of a Vector v:

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
    System.out.println(e.nextElement());
}
```

Methods are provided to enumerate through the elements of a vector (I-§3.10.13), the keys of a hash table (I-§3.4.11), and the values in a hash table (I-§3.4.8). Enumerations are also used to specify the input streams to a SequenceInputStream (I-§2.21)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Enumeration.hasMoreElements

public abstract boolean hasMoreElements ()

Returns:

true if this enumeration contains more elements; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Enumeration.nextElement

public abstract Object `nextElement()`

Returns:

the next element of this enumeration.

Throws

NoSuchElementException ([I-§3.14](#))

If no more elements exist.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.12 Interface Observer

```
public interface java.util.Observer
{
    // Methods
    public abstract void update(Observable o, Object arg);    §3.12.1
}
```

A class can implement the Observer interface when it wants to be informed if observable (I-§3.5) objects change.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Observer.update

public abstract void update(Observable o, Object arg)

This method is called whenever the observed object is changed. An application calls an observable object's notifyObservers method (§3.5.8, §3.5.9) to have all the object's observers notified of the change.

Parameters:

o- the observable object

arg- an argument passed to the notifyObservers method

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.13 Class EmptyStackException

```
public class java.util.EmptyStackException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public EmptyStackException(); §3.13.1
}
```

Thrown by methods in the Stack class (I-§3.8) to indicate that the stack is empty.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


EmptyStackException.EmptyStackException

public **EmptyStackException**()

Constructs a new EmptyStackException with no detail message.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.14 Class NoSuchElementException

```
public class java.util.NoSuchElementException
    extends java.lang.RuntimeException (I-§1.42)
{
    // Constructors
    public NoSuchElementException(); §3.14.1
    public NoSuchElementException(String s); §3.14.2
}
```

Thrown by the `nextElement` method (I-§3.11.2) of an Enumeration to indicate that there are no more elements in the enumeration.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NoSuchElementException.NoSuchElementException

public NoSuchElementException()

Constructs a NoSuchElementException with no detail message.

public NoSuchElementException(String s)

Constructs a NoSuchElementException with the specified detail message.

Parameters:

s- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.net

Classes

- 4.1 Class ContentHandler
- 4.2 Class DatagramPacket
- 4.3 Class DatagramSocket
- 4.4 Class InetAddress
- 4.5 Class ServerSocket
- 4.6 Class Socket
- 4.7 Class SocketImpl
- 4.8 Class URL
- 4.9 Class URLConnection
- 4.10 Class URLEncoder
- 4.11 Class URLStreamHandler

Interfaces

- 4.12 Interface ContentHandlerFactory
- 4.13 Interface SocketImplFactory
- 4.14 Interface URLStreamHandlerFactory

Exceptions

- 4.15 Class MalformedURLException
- 4.16 Class ProtocolException
- 4.17 Class SocketException
- 4.18 Class UnknownHostException
- 4.19 Class UnknownServiceException

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.1 Class ContentHandler

```
public abstract class java.net.ContentHandler
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public ContentHandler() ; §4.1.1

    // Methods
    public abstract Object getContent(URLConnection urlc) ; §4.1.2
}
```

The abstract class ContentHandler is the superclass of all classes that reads an Object from a URLConnection (I-§4.9).

An application does not generally call the getContentHandler method (I-§4.1.2) in this class directly. Instead, an application calls the getContent method in class URL (I-§4.8.6) or in URLConnection (I-§4.9.11). The application's content handler factory (an instance of a class that implements the interface ContentHandlerFactory (I-§4.12) set up by a call to setContentHandler (I-§4.9.37)) is called with a String giving the MIME type of the object being received on the socket. The factory returns an instance of a subclass of ContentHandler, and its getContent method is called to create the object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ContentHandler.ContentHandler

public ContentHandler ()

The default constructor for class ContentHandler.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ContentHandler.getContent

public abstract Object getContent(URLConnection urlc)
throws IOException

Given an URL connect stream positioned at the beginning of the representation of an object, this method reads that stream and creates an object from it.

Parameters:

`urlc`- a URL connection

Returns:

The object read by the ContentHandler.

Throws

IOException ([I-§2.29](#))

If an IO error occurs while reading the object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.2 Class DatagramPacket

```
public final class java.net.DatagramPacket
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public DatagramPacket(byte ibuf[], int ilength); §4.2.1
    public DatagramPacket(byte ibuf[], int ilength, §4.2.2
        InetAddress iaddr, int ippot);

    // Methods
    public InetAddress getAddress(); §4.2.3
    public byte[] getData(); §4.2.4
    public int getLength(); §4.2.5
    public int getPort(); §4.2.6
}
```

This class represents a datagram packet.

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from a machine to another might be routed differently, and might arrive in any order.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DatagramPacket.DatagramPacket

public DatagramPacket(byte ibuf[], int ilength)

Constructs a DatagramPacket for receiving packets of length ilength.

The length argument must be less than or equal to ibuf.length.

Parameters:

ibuf- buffer for holding the incoming datagram

ilength- the number of bytes to read

public DatagramPacket(byte ibuf[], int ilength, InetAddress iaddr, int iport)

Constructs a DatagramPacket for sending packets of length ilength to the specified port number on the specified host.

The length argument must be less than or equal to ibuf.length.

Parameters:

ibuf- the packet data

ilength- the packet length

iaddr- the desination address (\$4.4)

iport- the destination port number

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramPacket.getAddress

public InetAddress getAddress ()

Returns:

the IP address (\$4.4) of the machine to which this datagram is being sent, or from which the datagram was received.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramPacket.getData

public byte[] `getData()`

Returns:

the data received, or the data to be sent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramPacket.getLength

public int getLength()

Returns:

the length of the data to be sent, or the length of the data received.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramPacket.getPort

public int getPort()

Returns:

the port number on the remote host to which this datagram is being sent, or from which the datagram was received.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.3 Class DatagramSocket

```
public class java.net.DatagramSocket
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public DatagramSocket(); §4.3.1
    public DatagramSocket(int port); §4.3.2

    // Methods
    public void close(); §4.3.3
    protected void finalize(); §4.3.4
    public int getLocalPort(); §4.3.5
    public void receive(DatagramPacket p); §4.3.6
    public void send(DatagramPacket p); §4.3.7
}
```

This class represents a socket for sending and receiving datagram packets (§4.2).

A datagram socket is the sending or receiving point for a connectionless packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from a machine to another may be routed differently, and may arrive in any order.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramSocket.DatagramSocket

public DatagramSocket()
throws SocketException

Constructs a datagram socket and binds it to any available port on the local host machine.

Throws

SocketException ([I-§4.17](#))

If the socket could not be opened, or the socket could not bind the specified local port.

public DatagramSocket(int port)
throws SocketException

Constructs a datagram socket and binds it to the specified port on the local host machine.

Parameters:

local- port to use

Throws

SocketException ([I-§4.17](#))

If the socket could not be opened, or the socket could not bind the specified local port.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramSocket.close

public void close()

Closes this datagram socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramSocket.finalize

protected void finalize()

Ensures that this socket is closed if there are no longer any references to this socket.

Overrides:

finalize in class Object ([I-§1.12.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramSocket.getLocalPort

```
public int getLocalPort()
```

Returns:

the port number on the local host to which this socket is bound.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DatagramSocket.receive

public void receive(DatagramPacket p)
throws IOException

Receives a datagram packet from this socket. When this method returns, the DatagramPacket's buffer is filled with the data received. The datagram packet also contains the sender's IP address, and the port number on the sender's machine.

This method blocks until a datagram is received. The length field of the datagram packet object contains the length of the received message. If the message is longer than the buffer length, the message is truncated.

Parameters:

p- the DatagramPacket into which to place the incoming data

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DatagramSocket.send

public void send(DatagramPacket p)
throws IOException

Sends a datagram packet from this socket. The DatagramPacket ([I-§4.2](#)) includes information indicating the data to be sent, its length, the IP address of the remote host, and the port number on the remote host.

Parameters:

p- the DatagramPacket to be sent

Throws

IOException ([I-§2.29](#))

If an I/O error occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.4 Class InetAddress

```
public final class java.net.InetAddress
    extends java.lang.Object (I-§1.12)
{
    // Methods
    public boolean equals(Object obj); §4.4.1
    public byte[] getAddress(); §4.4.2
    public static InetAddress[] getAllByName(String host); §4.4.3
    public static InetAddress getByName(String host); §4.4.4
    public String getHostName(); §4.4.5
    public static InetAddress getLocalHost(); §4.4.6
    public int hashCode(); §4.4.7
    public String toString(); §4.4.8
}
```

This class represents an Internet Protocol (IP) address.

Applications should use the methods `getLocalHost` (I-§4.4.6), `getByName` (I-§4.4.4), or `getAllByName` (I-§4.4.3) to create a new `InetAddress` instance.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InetAddress.equals

public boolean equals (Object obj)

The result is true if the argument is not null and it represents the same IP address as this object.

Two instances of InetAddress represent the same IP address if the length of the byte arrays returned by getAddress ([I-§4.4.2](#)) is the same for both, and each of the array components is the same for the byte arrays.

Parameters:

obj – the object to compare against

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.getAddress

public byte[] getAddress()

Determines the raw IP address of this InetAddress object. The result is in network byte order; the highest order byte of the address is in getAddress()[0].

Returns:

the raw IP address of this object.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.getAllByName

```
public static InetAddress[] getAllByName(String host)  
throws UnknownHostException
```

Parameters:

`host`– the name of the host

Returns:

an array of all the IP addresses for a given host name.

Throws

UnknownHostException ([I-§4.18](#))

If no IP address for the host could be found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


InetAddress.getByName

public static InetAddress getByName(String host)
throws UnknownHostException

Determines the IP address of a host, given the host's name. The host name can either be a machine name, such as "java.sun.com" or a string representing its IP address, such as "206.26.48.100."

Parameters:

`host` - the specified host, or null for the local host

Returns:

an IP address for the given host name.

Throws

UnknownHostException (I-§4.18)

If no IP address for the host could be found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.getHostName

public String getHostName()

Returns:

the host name for this IP address.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.getLocalHost

`public static InetAddress getLocalHost()
throws UnknownHostException`

Returns:

the IP address of the local host.

Throws

UnknownHostException (I-§4.18)

If no IP address for the host could be found.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.hashCode

public int hashCode()

Returns:

a hash code value for this IP address.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

InetAddress.toString

public String toString()

Returns:

a string representation of this IP address.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.5 Class ServerSocket

```
public final class java.net.ServerSocket
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public ServerSocket(int port); §4.5.1
    public ServerSocket(int port, int count); §4.5.2

    // Methods
    public Socket accept(); §4.5.3
    public void close(); §4.5.4
    public InetAddress getInetAddress(); §4.5.5
    public int getLocalPort(); §4.5.6
    public static void §4.5.7
        setSocketFactory(SocketImplFactory fac);
    public String toString(); §4.5.8
}
```

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

The actual work of the server socket is performed by an instance of the SocketImpl class (l-§4.7). An application can change the socket factory that creates the socket implementation (l-§4.5.7) to configure itself to create sockets appropriate to the local firewall.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ServerSocket.ServerSocket

public ServerSocket(int port)
throws IOException

Creates a server socket on a specified port. A port of 0 creates a socket on any free port.

The maximum queue length for incoming connection indications (a request to connect) is set to 50. If a connection indication arrives when the queue is full, the connection is refused.

If the application has specified a server socket factory (I-§4.5.7), that factory's createSocketImpl method (I-§4.13.1) is called to create the actual socket implementation. Otherwise a "plain" socket (see I-§4.7) is created.

Parameters:

port- the port number, or 0 to use any free port

Throws

IOException (I-§2.29)

If an IO error occurs when opening the socket.

public ServerSocket(int port, int count)
throws IOException

Creates a server socket and binds it to the specified local port number. A port number of 0 creates a socket on any free port.

The maximum queue length for incoming connection indications (a request to connect) is set to the count parameter. If a connection indication arrives when the queue is full, the connection is refused.

If the application has specified a server socket factory (I-§4.5.7), that factory's createSocketImpl method (I-§4.13.1) is called to create the actual socket implementation. Otherwise a "plain" socket (see I-§4.7) is created.

Parameters:

`port`– the specified port, or 0 to use any free port

`count`– the maximum length of the queue

Throws

IOException (I-§2.29)

if an I/O error occurs when opening the socket.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ServerSocket.accept

public Socket accept()
throws IOException

Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when waiting for a connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ServerSocket.close

public void close()
throws IOException

Closes this socket.

Throws

IOException ([I-§2.29](#))

If an I/O occurs error when closing the socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ServerSocket.getInetAddress

public InetAddress getInetAddress ()

Returns:

The address to which this socket is connected, or null if the socket is not yet connected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ServerSocket.getLocalPort

public int getLocalPort()

Returns:

the port number to which this socket is listening.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ServerSocket.setSocketFactory

public static void setSocketFactory(SocketImplFactory fac)
throws IOException

Sets the server socket implementation factory for the application. The factory can be specified only once.

When an application creates a new server socket, the socket implementation factory's `createSocketImpl` method (I-§4.13.1) is called to create the actual socket implementation.

Parameters:

`fac` – the desired factory

Throws

SocketException (I-§4.17)

If the factory has already been defined.

Throws

IOException (I-§2.29)

If an I/O error occurs when setting the socket factory.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ServerSocket.toString

public String toString()

Returns:

a string representation of this socket.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.6 Class Socket

```
public final class java.net.Socket
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public Socket(InetAddress address, int port); §4.6.1
    public Socket(InetAddress address, int port, §4.6.2
                    boolean stream);
    public Socket(String host, int port); §4.6.3
    public Socket(String host, int port, boolean stream); §4.6.4

    // Methods
    public void close(); §4.6.5
    public InetAddress getInetAddress(); §4.6.6
    public InputStream getInputStream(); §4.6.7
    public int getLocalPort(); §4.6.8
    public OutputStream getOutputStream(); §4.6.9
    public int getPort(); §4.6.10
    public static void §4.6.11
        setSocketImplFactory(SocketImplFactory fac);
    public String toString(); §4.6.12
}
```

This class implements client sockets (also called just "sockets"). A socket is a end point for communication between two machines.

The actual work of the socket is performed by an instance of the SocketImpl class (l-§4.7). An application, by changing the socket factory that creates the socket implementation (l-§4.6.11), can configure itself to create sockets appropriate to the local firewall.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Socket.Socket

public Socket(InetAddress address, int port)
throws IOException

Creates a stream socket and connects it to the specified port number at the specified IP address.

If the application has specified a socket factory ([I-§4.6.11](#)), that factory's createSocketImpl method ([I-§4.13.1](#)) is called to create the actual socket implementation. Otherwise a "plain" socket (see [I-§4.7](#)) is created.

Parameters:

address- the IP address

port- the port number

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the socket.

public Socket(InetAddress address, int port, boolean stream)
throws IOException

Creates a socket and connects it to the specified port number at the specified IP address.

If the stream [argument](#) is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a [server](#) socket factory ([I-§4.6.11](#)), that factory's createSocketImpl method ([I-§4.13.1](#)) is called to create the actual socket implementation. Otherwise a "plain" socket (see [I-§4.7](#)) is created.

Parameters:

address- the IP address

port- the port number

stream- if true, create a stream socket; if false, create a datagram socket

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the socket.

public Socket(String host, int port)
throws UnknownHostException, IOException

Creates a stream socket and connects it to the specified port number on the named host.

If the application has specified a server socket factory ([I-§4.6.11](#)), that factory's createSocketImpl [method](#) ([I-§4.13.1](#)) is called to create the actual socket implementation. Otherwise a "plain" socket (see [I-§4.7](#)) is created.

Parameters:

host- the host name

port- the port number

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the socket.

public Socket(String host, int port, boolean stream)
throws IOException

Creates a stream socket and connects it to the specified port number on the named host.

If the stream argument is true, this creates a stream socket. If the stream argument is false, it creates a datagram socket.

If the application has specified a server socket factory ([I-§4.6.11](#)), that factory's createSocketImpl [method](#) ([I-§4.13.1](#)) is called to create the actual socket implementation. Otherwise a "plain" socket (see [I-§4.7](#)) is created.

Parameters:

host- the host name

port- the port number

`stream`- a boolean indicating whether this is a stream or datagram socket

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the socket.

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

Socket.close

public void close()
throws IOException

Closes this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when closing this socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Socket.getInetAddress

public InetAddress getInetAddress ()

Returns:

the remote IP address to which this socket is connected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Socket.getInputStream

```
public InputStream getInputStream()  
throws IOException
```

Returns:

an input stream for reading bytes from this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Socket.getLocalPort

```
public int getLocalPort()
```

Returns:

the local port number to which this socket is connected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Socket.getOutputStream

`public OutputStream getOutputStream()
throws IOException`

Returns:

an output stream for writing bytes to this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Socket.getPort

```
public int getPort()
```

Returns:

the remote port number to which this socket is connected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Socket.setSocketImplFactory

public static void

setSocketImplFactory(SocketImplFactory fac)
throws IOException

Sets the client socket implementation factory for the application. The factory can be specified only once.

When an application creates a new client socket, the socket implementation factory's `createSocketImpl` [method \(I-§4.13.1\)](#) is called to create the actual socket implementation.

Parameters:

`fac` – the desired factory

Throws

SocketException [\(I-§4.17\)](#)

If the factory is already defined.

Throws

IOException [\(I-§2.29\)](#)

If an I/O error occurs when setting the socket factory.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Socket.toString

public String toString()

Returns:

a string representation of this socket.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.7 Class SocketImpl

```
public abstract class java.net.SocketImpl
    extends java.lang.Object (I-§1.12)
{
    // Fields
    protected InetAddress address; §4.7.1
    protected FileDescriptor fd; §4.7.2
    protected int localport; §4.7.3
    protected int port; §4.7.4

    // Constructors
    public SocketImpl(); §4.7.5

    // Methods
    protected abstract void accept(SocketImpl s); §4.7.6
    protected abstract int available(); §4.7.7
    protected abstract void bind(InetAddress host, int port); §4.7.8
    protected abstract void close(); §4.7.9
    protected abstract void §4.7.10
        connect(InetAddress address, int port);
    protected abstract void connect(String host, int port); §4.7.11
    protected abstract void create(boolean stream); §4.7.12
    protected FileDescriptor getFileDescriptor(); §4.7.13
    protected InetAddress getInetAddress(); §4.7.14
    protected abstract InputStream getInputStream(); §4.7.15
    protected int getLocalPort(); §4.7.16
    protected abstract OutputStream getOutputStream(); §4.7.17
    protected int getPort(); §4.7.18
    protected abstract void listen(int count); §4.7.19
    public String toString(); §4.7.20
}
```

The abstract class `SocketImpl` is a common superclass of all classes that actually implement sockets. It is used to create both client and server sockets.

A "plain" socket implements these methods exactly as described, without attempting to go through a firewall or proxy.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SocketImpl.address

protected **InetAddress** address

The IP address of the remote end of this socket.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SocketImpl.fd

protected FileDescriptor fd

The file descriptor object for this socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.localport

protected int localport

The local port number to which this socket is connected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.port

protected int port

The port number on the remote host to which this socket is connected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.SocketImpl

public `SocketImpl()`

The default constructor for a socket implementation.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.accept

protected abstract void accept(SocketImpl s)
throws IOException

Accepts a connection.

Parameters:

s- the accepted connection

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when accepting the connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.available

protected abstract int available()
throws IOException

Returns:

the number of bytes that can be read from this socket without blocking.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when determining the number of bytes available.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.bind

protected abstract void bind(InetAddress host, int port)
throws IOException

Binds this socket to the specified port number on the specified host.

Parameters:

host- the IP address of the remote host

port- the port number

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when binding this socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.close

protected abstract void close()
throws IOException

Closes this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when closing this socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.connect

protected abstract void

connect(InetAddress address, int port)
throws IOException

Connects this socket to the specified port number on the specified host.

Parameters:

address- the IP address of the remote host

port- the port number

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when attempting a connection.

protected abstract void connect(String host, int port)
throws IOException

Connects this socket to the specified port on the named host.

Parameters:

host- the name of the remote host

port- the port number

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when connecting to the remote host.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.create

protected abstract void create(boolean stream)
throws IOException

Creates a socket.

Parameters:

`stream`- if true, creates a stream socket; otherwise, creates a datagram socket

Throws

IOException ([I-§2.29](#))

If an IO error occurs while creating the socket.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getFileDescriptor

protected FileDescriptor getFileDescriptor()

Returns:

the value of this socket's fd field (I-§4.7.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getInetAddress

protected InetAddress getInetAddress ()

Returns:

the value of this socket's address field (I-§4.7.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getInputStream

protected abstract InputStream getInputStream()
throws IOException

Returns:

a stream for reading from this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getLocalPort

protected int getLocalPort()

Returns:

the value of this socket's LocalPort field (I-§4.7.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getOutputStream

protected abstract OutputStream getOutputStream()
throws IOException

Returns:

an output stream for writing to this socket.

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the output stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.getPort

protected int getPort()

Returns:

the value of this socket's port field (~~I-§4.7.4~~).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.listen

protected abstract void listen(int count)
throws IOException

Sets the maximum queue length for incoming requests to this socket to the count argument.
If a connection request arrives when the queue is full, the connection is refused.

Parameters:

count- the maximum length of the queue

Throws

IOException ([I-§2.29](#))

If an I/O error occurs when creating the queue.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

SocketImpl.toString

public String toString()

Returns:

a string representation of this socket.

Overrides:

toString in class Object ([1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.8 Class URL

```
public final class java.net.URL
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public URL(String spec); §4.8.1
    public URL(String protocol, String host, §4.8.2
                int port, String file);
    public URL(String protocol, String host, String file); §4.8.3
    public URL(URL context, String spec); §4.8.4

    // Methods
    public boolean equals(Object obj); §4.8.5
    public final Object getContent(); §4.8.6
    public String getFile(); §4.8.7
    public String getHost(); §4.8.8
    public int getPort(); §4.8.9
    public String getProtocol(); §4.8.10
    public String getRef(); §4.8.11
    public int hashCode(); §4.8.12
    public URLConnection openConnection(); §4.8.13
    public final InputStream openStream(); §4.8.14
    public boolean sameFile(URL other); §4.8.15
    public static void §4.8.16
        setURLStreamHandlerFactory(URLStreamHandlerFactory fac);
    public String toExternalForm(); §4.8.17
    public String toString(); §4.8.18
}
```

Class URL represents a Uniform Resource Locator—a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine. More information on the types of URLs and their format can be found at:

<http://www.ncsa.uiuc.edu/demoweb/url-primer.html>

In general, an URL can be broken into several parts. The above URL indicates that the protocol to use is http ("HyperText Transport Protocol"), that the information resides on a host whose name is www.ncsa.uiuc.edu. The information on that host machine is named demoweb/url-primer.html. The exact meaning of its name on the host machine is both protocol- and host-dependent. The information could reside in a file or could be generated on-the-fly.

A URL can optionally contain a "port," which is the port number to which the connection is made on the remote host. If the port is not specified, the default port for the URL is used instead. For example, the default port for http, is 80. An alternative port could be specified as:

`http://www.ncsa.uiuc.edu:8080/demoweb/url-primer.html`

A URL may have appended to it an "anchor", which is indicated by the sharp sign character "#" followed by more characters. For example

`http://local/demo/information#myinfo`

This is not technically part of the URL. Rather, it indicates that after the specified "resource" is retrieved, the application is specifically interested in that part of the document that has the tag "myinfo" attached to it. The meaning of a tag is resource specific.

An application can also specify a "relative URL", which contains only enough information to reach the resource relative to another URL. Relative URLs are frequently used within HTML pages. For example, if the URL

`http://java.sun.com/index.html`

contained within it a reference to the URL "FAQ.html", it would be a shorthand for

`http://java/sun.com/FAQ.html`

The relative URL need not specify all the components of a URL. Missing components are inherited from the fully specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.URL

public URL(String spec)
throws MalformedURLException

Creates a URL object from the String representation.

This constructor is equivalent to a call to the two-argument constructor (I-§4.8.4) with a null first argument.

Parameters:

spec- the String to parse as a URL

Throws

MalformedURLException (I-§4.15)

If the string specifies an unknown protocol.

public URL(String protocol, String host, int port, String file)
throws MalformedURLException

Creates a URL object from the specified protocol, host, port number, and file.

If this is the first URL object being created with the specified protocol, a stream protocol handler object, an instance of class URL-Stream-Handler (I-§4.11), is created for that protocol.

If the application has previously set up an instance of URL-Stream-Handler-Factory as the stream handler factory (I-§4.8.16), then the createURLStreamHandler (I-§4.14.1) method of that instance is called with the protocol string as an argument to create the stream protocol handler.

If no URLStreamHandlerFactory has yet been set up, or if the factory's createURLStreamHandler method returns null, then the constructor finds the value of the system property (I-§1.18.9) java.handler.protol.pkgs.

If the value of that system property is not null, it is interpreted as a list of packages separated by a vertical slash character '|'. The constructor tries to load the class named

<package>.<protocol>.Handler

where <package> is replaced by the name of the package and <protocol> is replaced by the name of the protocol. If this class does not exist, or if it the class exists but it is not a subclass of URLStreamHandler, then the next package in the list is tried.1

If the previous step fails to find a protocol handler, then the constructor tries to load the class named `sun.net.www.protocol.<protocol>.Handler`. If this class does not exist, or if the class exists but it is not a subclass of `URLConnectionHandler`, then a `MalformedURLException` is thrown.

Parameters:

`protocol`- the name of the protocol
`host`- the name of the host
`port`- the port number
`file`- the name of the information

Throws

`MalformedURLException` (I-§4.15)
if an unknown protocol is specified.

`public URL(String protocol, String host, String file)`
`throws MalformedURLException`

Creates an absolute URL from the specified protocol name, host name, and file name. The default port for the specified protocol is used.

The constructor searches for an appropriate `URLConnectionHandler` (I-§4.11) as outlined in I-§4.8.2.

Parameters:

`protocol`- the protocol to use
`host`- the host to connect to
`file`- the name of the information

Throws

`MalformedURLException` (I-§4.15)
if an unknown protocol is specified.

`public URL(URL context, String spec)`
`throws MalformedURLException`

Creates a URL by parsing the String specification within a specified context. If the context argument is not null and the spec argument is a partial URL specification, then any of the strings' missing components are inherited from the context argument.

The specification given by the String argument is parsed to determine if it specifies a protocol. If the String contains an ASCII colon ':' character before the first occurrence of an ASCII slash character '/', then the characters before the colon comprise the protocol.

- If the spec argument does not specify a protocol:
 - If the context argument isn't null, then the protocol is copied from the context argument.
 - If the context argument is null, then a MalformedURLException is thrown.
- If the spec argument does specify a protocol:

If the context argument is null, or specifies a different protocol than the specification argument, the context argument is ignored.

If the context argument isn't null and specifies the same protocol as the specification, the host, port number, and file are copied from the context argument into the newly created URL.

The constructor then searches for an appropriate stream protocol handler of type URLStreamHandler (I-§4.11) as outlined in I-§4.8.2. The stream protocol handler's parseURL method (I-§4.11.3) is called to parse the remaining fields of the specification that override any defaults set by the context argument.

Parameters:

`context`– the context in which to parse the specification

`spec`– a String representation of a URL

Throws

MalformedURLException (I-§4.15)

If no protocol is specified, or an unknown protocol is found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a URL object that represents the same URL as this object. Two URL objects are equal if they have the same protocol, reference the same host, the same port number on the host, and the same information on the host.

Parameters:

obj – the URL to compare against

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.getContent

public final Object getContent()
throws IOException

Determines the contents of this URL. This method is a shorthand for:

```
openConnection().getContent()
```

Returns:

the contents of this URL.

Throws

IOException ([I-§2.29](#))

if an I/O exception occurs.

See Also:

getContent in class URLConnection ([I-§4.9.11](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

URL.getFile

```
public String getFile()
```

Returns:

the information field of this URL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.getHost

```
public String getHost()
```

Returns:

the host name field of this URL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.getPort

```
public int getPort()
```

Returns:

the port number of this URL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.getProtocol

```
public String getProtocol()
```

Returns:

the name of the protocol of this URL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.getRef

```
public String getRef()
```

Returns:

the anchor (also known as the "reference") of this URL.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.hashCode

public int hashCode()

Returns:

a hash code for this URL.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.openConnection

public **URLConnection** **openConnection()**
throws **IOException**

Creates (if not already in existence) a **URLConnection** object that represents a connection to the remote object referred to by the URL.

The connection is opened by calling the **openConnection** method (I-§4.11.2) of the protocol handler (I-§4.8.2) for this URL.

Returns:

a **URLConnection** (I-§4.9) to the URL.

Throws

IOException (I-§2.29)

If an I/O exception occurs.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.openStream

public final InputStream openStream()
throws IOException

Opens a connection to this URL and returns a stream for reading from that connection. This method is a shorthand for

```
openConnection().getInputStream()
```

Returns:

a stream for reading from the URL connection.

Throws

IOException ([I-§2.29](#))

If an I/O exception occurs.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.sameFile

public boolean sameFile(URL other)

Returns true if the this URL and the other argument both refer to the same resource; the two URLs might not both contain the same anchor.

Parameters:

`other`– the URL to compare against

Returns:

true if they reference the same remote object; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.setURLStreamHandlerFactory

public static void

setURLStreamHandlerFactory(URLStreamHandlerFactory fac)

Sets an application's URLStreamHandlerFactory (I-§4.14). This method can be called at most once by an application.

The URLStreamHandlerFactory instance is used to construct a stream protocol handler (I-§4.8.2) from a protocol name.

Parameters:

fac- the desired factory

Throws

Error (I-§1.48)

If the application has already set a factory.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URL.toExternalForm

public String toExternalForm()

Constructs a string representation of this URL. The string is created by calling the toExternalForm method (I-§4.11.5) of the stream protocol handler (I-§4.8.2) for this object.

Returns:

a string representation of this object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URL.toString

public String toString()

Creates a string representation of this object. This method calls the toExternalForm method (I-§4.8.17) and returns its value.

Returns:

a string representation of this object.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹Step 2 is new in Java 1.1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§4.9 Class URLConnection

```
public abstract class java.net.URLConnection
    extends java.lang.Object (l-§1.12)
{
    // Fields
    protected boolean allowUserInteraction;    §4.9.1
    protected boolean connected;               §4.9.2
    protected boolean doInput;                 §4.9.3
    protected boolean doOutput;                §4.9.4
    protected long   ifModifiedSince;           §4.9.5
    protected URL    url;                      §4.9.6
    protected boolean useCaches;                §4.9.7

    // Constructors
    protected URLConnection(URL url);          §4.9.8

    // Methods
    public abstract void connect();             §4.9.9
    public boolean getAllowUserInteraction();    §4.9.10
    public Object getContent();                 §4.9.11
    public String getContentEncoding();          §4.9.12
    public int   getContentLength();             §4.9.13
    public String getContentType();              §4.9.14
    public long  getDate();                      §4.9.15
    public static boolean getDefaultAllowUserInteraction(); §4.9.16
    public static String §4.9.17
        getDefaultRequestProperty(String key);
    public boolean getDefaultUseCaches();        §4.9.18
    public boolean getDoInput();                 §4.9.19
    public boolean getDoOutput();                §4.9.20
    public long   getExpiration();                §4.9.21
    public String getHeaderField(int n);          §4.9.22
    public String getHeaderField(String name);    §4.9.23
    public long   getHeaderFieldDate(String name, long Default); §4.9.24
    public int    getHeaderFieldInt(String name, int Default);   §4.9.25
    public String getHeaderFieldKey(int n);       §4.9.26
    public long   getIfModifiedSince();           §4.9.27
    public InputStream getInputStream();          §4.9.28
    public long   getLastModified();              §4.9.29
    public OutputStream getOutputStream();        §4.9.30
    public String getRequestProperty(String key); §4.9.31
    public URL getURL();                          §4.9.32
    public boolean getUseCaches();                §4.9.33
    protected static String §4.9.34
        guessContentTypeFromName(String fname);
```



```

    protected static String §4.9.35
        guessContentTypeFromStream(InputStream is);
    public void §4.9.36
        setAllowUserInteraction(boolean allowuserinteraction);
    public static void §4.9.37
        setContentHandlerFactory(ContentHandlerFactory fac);
    public static void §4.9.38
        setDefaultAllowUserInteraction(boolean defaultallowuserinteraction);
    public static void §4.9.39
        setDefaultRequestProperty(String key, String value);
    public void §4.9.40
        setDefaultUseCaches(boolean defaultusecaches);
    public void setDoInput(boolean doinput); §4.9.41
    public void setDoOutput(boolean dooutput); §4.9.42
    public void setIfModifiedSince(long ifmodifiedsince); §4.9.43
    public void setRequestProperty(String key, ;String value); §4.9.44
    public void setUseCaches(boolean usecaches); §4.9.45
    public String toString(); §4.9.46
}

```

The abstract class `URLConnection` is the superclass of all classes that represent a communications link between the application and a URL. Instances of this class can be used both to read from and to write to the resource referenced by the URL. In general, creating a connection to a URL is a multi-step process: {ewc msdncl, EWGraphic, NET0dm 0 /a "sunref.BMP"}

First, the connection object is created by invoking `openConnection` on a URL, then setup parameters are manipulated, then the actual connection to the remote object is made, and finally the remote object is available. There are a set of properties that apply before the connection is made, and a set that apply after:

before connect	after connect
AllowUserInteraction	ContentEncoding
RequestProperty	ContentLength
UseCaches	ContentType
DoOutput	HeaderField
DoInput	InputStream
IfModifiedSince	OutputStream
	LastModified
	Date
	LastModified
	Expiration

Most of these properties have a set and a get method. The first three in the "before" column also have corresponding "Default" properties that specify the default values used in subsequent requests where the property is not specified.

In the common case, all of these properties can be ignored: the pre-connection properties default to sensible values, and the post-connection properties are often uninteresting. For most clients of this interface, there are only two interesting methods: `getInputStream` and `getObject`, which are mirrored in the `URL` class by convenience methods.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.allowUserInteraction

protected boolean allowUserInteraction

If true, this URL is being examined in a context which it makes sense to allow user interactions such as popping up an authentication dialog. If false, then no user interaction is allowed.

This value of this field can be set by the set-Allow-User-Interaction method [\(I-§4.9.36\)](#). Its value can be accessed by the get-Allow-UserInteraction method [\(I-§4.9.10\)](#). Its default value is the value of the argument method to the last call to the set--Default-Allow-User-Interaction method [\(I-§4.9.38\)](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.connected

protected boolean connected

If false, this connection object has not created a communications link to the specified URL. If true, the communications link has been established.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.doInput

protected boolean doInput

This variable is set by the setDoInput method (I-§4.9.41). Its value can be accessed by the getDoInput method (I-§4.9.19).

A URL connection can be used for input and/or output. Setting the do-Input flag to true indicates that the application intends to read data from the URL connection.

The default value of this field is true.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.doOutput

protected boolean doOutput

This variable is set by the setDoOutput method ([I-§4.9.42](#)). Its value can be accessed by the getDoInput method ([I-§4.9.20](#)).

A URL connection can be used for input and/or output. Setting the do-Output flag to true indicates that the application intends to write data to the URL connection.

The default value of this field is false.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.ifModifiedSince

protected long ifModifiedSince

Some protocols support skipping the fetching of the object unless the object has been modified more recently than a certain time.

A non-zero value is interpreted as the time corresponding to if-Modified-Since seconds since January 1, 1970, GMT. The object is fetched only if it has been modified more recently than that time.

This variable is set by the setIfModifiedSince method (I-§4.9.43). Its value can be accessed by the getIfModifiedSince method (I-§4.9.27).

The default value of this field is 0, indicating that the fetching must always occur.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.url

protected URL url

The URL representing the remote object on the World Wide Web to which this connection is opened.

The value of this field can be accessed by the `getURL` method (I-§4.9.32).

The default value of this variable is the value of the URL argument in the `URLConnection` constructor (I-§4.9.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.useCaches

protected **boolean** useCaches

If true, the protocol is allowed to use caching whenever it can. If false, the protocol must always try to get a fresh copy of the object.

This field is set by the setUseCaches method (I-§4.9.45). Its value can be accessed by the getUseCaches method (I-§4.9.33). Its default value is the value of the last argument to the method set-Default-Use-Caches (I-§4.9.40).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.URLConnection

protected `URLConnection(URL url)`

Constructs a URL connection to the specified URL. A connection to the object referenced by the URL is not created.

Parameters:

`url`- the specified URL

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.connect

public abstract void connect()
throws IOException

Opens a communications link to the resource referenced by this URL, if such a connection hasn't already been established.

If the connect method is called when the connection has already been opened (indicated by the connected field (I-§4.9.2) having the value true), the call is ignored.

Throws

IOException (I-§2.29)

If an IO error occurs while opening the connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getAllowUserInteraction

public boolean getAllowUserInteraction()

Returns:

the value of the allowUserInteraction (I-§4.9.1)field for this object..

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getContent

public Object getContent()
throws IOException

Retrieves the contents of this URL connection.

This method first determines the content type of the object by calling the method `getContentType` ([I-§4.9.1](#)). If this is the first time that the application has seen that specific content type, a content handler for that content type is created:

If the application has set up a content handler factory instance using the `setContentHandlerFactory` ([I-§4.9.37](#)) method, the `createContentHandler` ([I-§4.12.1](#)) method of that instance is called with the content type as an argument; the result is a content handler for that content type.

If no content handler factory has yet been set up, or if the factory's `createContentHandler` method returns null, then the application loads the class named

`sun.net.www.content.<contentType>`

where `<contentType>` is formed by taking the content type string, replacing all slash characters with a period '.', and all other non-alphanumeric characters with the underscore character '_'. If the specified class does not exist, or is not a subclass of `ContentHandler`, then an `UnknownServiceException` is thrown.

Returns:

the object fetched. The instance of operation should be used to determine the specific kind of object returned.

Throws

IOException ([I-§2.29](#))

If an IO error occurs while getting the content.

Throws

UnknownServiceException ([I-§4.19](#))

If the protocol does not support the content type.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getContentEncoding

public String getContentEncoding()

Determines the value of the content-encoding attribute.

Returns:

the content encoding of the resource that the URL references, or null if not known.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getLength()

public int getLength()

Determines the value of the content-length attribute.

Returns:

the content length of the resource that this connection's URL references, or -1 if the content length is not known.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getContentType

public String getContentType()

Determines the value of the content-type attribute.

Returns:

the content type of the resource that the URL references, or null if not known.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDate

public long getDate()

Determines the value of the date attribute.

Returns:

the sending date of the resource that the URL references, or 0 if not known. The value returned is the number of seconds since January 1, 1970 GMT.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDefaultAllowUserInteraction

public static boolean getDefaultAllowUserInteraction()

Returns:

The default value of the allowUserInteraction (I-§4.9.1)field.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDefaultRequestProperty

public static String getDefaultRequestProperty(String key)

Gets the value of the default request property. Default request properties are set for every connection.

Returns:

the value of the default request property for the specified key.

See Also:

setDefaultRequestProperty ([1-§4.9.39](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDefaultUseCaches

public boolean getDefaultUseCaches ()

Returns:

the default value of this URLConnection's useCaches (I-§4.9.7) flag.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDoInput

public boolean getDoInput()

Returns:

the value of this URLConnection's doInput (I-§4.9.3)flag.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getDoOutput

public boolean getDoOutput ()

Returns:

the value of this URLConnection's doOutput (I-§4.9.4)flag.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getExpiration

public long getExpiration()

Determines the value of the expires attribute.

Returns:

the expiration date of the resource that this URL references, or 0 if not known. The value is number of seconds since January 1, 1970 GMT.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getHeaderField

`public String getHeaderField(int n)`

Gets the value for the nth header field. It returns null if there are fewer than n fields.

This method can be used in conjunction with the `getHeaderFieldKey` method (I-§4.9.26) to iterate through all the headers in the message.

Parameters:

`n`— an index

Returns:

the value of the nth header field.

`public String getHeaderField(String name)`

Parameters:

`name`— the name of a header field

Returns:

the value of the named header field, or null if not known.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getHeaderFieldDate

public long getHeaderFieldDate(String name, long Default)

The named header field is parsed as a date. The result is the number of seconds since January 1, 1970 GMT represented by the named field.

This form of getHeaderField exists because some connection types (e.g. http-ng) have pre-parsed headers. Classes for that connection type can override this method and short-circuit the parsing.

Parameters:

name- the name of the header field

Default- a default value

Returns:

the value of the field, parsed as a date. The value of the Default argument is returned if the field is missing or malformed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getHeaderFieldInt

```
public int getHeaderFieldInt(String name, int Default)
```

The named header field is parsed as a number. The value is returned.

This form of getHeaderField exists because some connection types (e.g. http-ng) have pre-parsed headers. Classes for that connection type can override this method and short-circuit the parsing.

Parameters:

`name`– the name of the header field

`Default`– the default value

Returns:

the value of the named field, parsed as an integer. The Default value is returned if the field is missing or malformed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.getHeaderFieldKey

public String getHeaderFieldKey(int n)

Parameters:

n- an index

Returns:

the key for the nth header field, or null if there are fewer than n fields.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.getIfModifiedSince

public long getIfModifiedSince()

Returns:

the value of this object's ifModifiedSince (I-§4.9.5)field.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getInputStream

`public InputStream getInputStream()
throws IOException`

Returns:

an input stream that reads from this open connection.

Throws

IOException ([I-§2.29](#))

If an IO error occurs while creating the input stream.

Throws

UnknownServiceException ([I-§4.19](#))

If the protocol does not support input.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getLastModified

public long getLastModified()

Determines the value of the last-modified attribute.

Returns:

the date the resource referenced by this URLConnection was last modified, or 0 if not known.
The result is the number of seconds since January 1, 1970 GMT.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getOutputStream

public OutputStream **getOutputStream()**
throws IOException

Returns:

an output stream that writes to this connection.

Throws

IOException ([I-§2.29](#))

If an IO error occurs while creating the output stream.

Throws

UnknownServiceException ([I-§4.19](#))

If the protocol does not support output.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getRequestProperty

```
public String getRequestProperty(String key)
```

Returns:

The value of the named general request property for this connection.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLConnection.getURL

public URL getURL ()

Returns:

the value of this URLConnection's URL (I-§4.9.6)field.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.getUseCaches

public boolean getUseCaches ()

Returns:

the value of this URLConnection's useCaches (I-§4.9.7)field

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.guessContentTypeFromName

protected static String guessContentTypeFromName(String fname)

Makes a guess as to what the content type of an object is, based upon the argument, which should be the "information" component of a URL. This is a convenience method which can be used by subclasses which override the getContentType method (I-§4.9.14).

Parameters:

fname- a file name

Returns:

a guess as to what the content type of the object is, based upon its name.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.guessContentTypeFromStream

protected static String guessContentTypeFromStream(InputStream is)
throws IOException

Tries to determine the type of an input stream based on the characters at the beginning of the input stream. This is a convenience method which can be used by subclasses which override the `getContentType` method ([I-§4.9.14](#))

Ideally, this routine would not be needed. But many http servers return the incorrect content type; in addition, there are many non-standard extensions. Direct inspection of the bytes to determine the content-type is often more accurate than believing the content-type claimed by the http server.

Parameters:

`is` - an input stream that supports marks.(that is, it must be built on top of a `BufferedInputStream`)

Returns:

a guess at the content type, or null if none can be determined.

Throws

IOException ([I-§2.29](#))

If an IO error occurs while reading the input stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setAllowUserInteraction

public void

setAllowUserInteraction(boolean allowuserinteraction) Set the value of the allowUserInteraction (I-§4.9.1) field of this URLConnection.

Parameters:

allowuserinteraction-
the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setContentHandlerFactory

public static void

setContentHandlerFactory(ContentHandlerFactory fac)

Sets the ContentHandlerFactory [\(I-§4.12\)](#) of an application. It can be called at most once by an application.

The ContentHandlerFactory instance is used to construct a content handler [\(I-§4.9.11\)](#) from a content type

Parameters:

fac – the desired factory

Throws

Error [\(I-§1.48\)](#)

If the factory has already been defined.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setDefaultAllowUserInteraction

```
public static void setDefaultAllowUserInteraction(  
    boolean defaultallowuserinteraction) Sets the default value  
of the allow-User-Interaction (I-§4.9.1)flag for this URLConnection to the specified value.
```

Parameters:

defaultallowuserinteraction-
the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setDefaultRequestProperty

public static void

setDefaultRequestProperty(String key, String value)

Sets the default value of a general request property. When a URL-Connection is created, it is initialized with these properties.

Parameters:

`key`- the keyword by which the request is known (eg "accept")

`value`- the value associated with the key.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setDefaultUseCaches

public void setDefaultUseCaches (boolean defaultusecaches)

Sets the default value of the useCaches (I-§4.9.7)flag for this URLConnection to the specified value.

Parameters:

defaultusecaches-

the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setDoInput

public void setDoInput(boolean doinput)

Sets the value of the doInput (I-§4.9.3)flag for this URLConnection to the specified value.

Parameters:

value- the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setDoOutput

public void setDoOutput(boolean dooutput)

Sets the value of the doOutput (I-§4.9.4)flag for this URL-Connection to the specified value.

Parameters:

value- the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setIfModifiedSince

public void setIfModifiedSince(long ifmodifiedsince)

Sets the value of the ifModifiedSince (I-§4.9.5) field of this URLConnection to the specified value

Parameters:

value- the new value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setRequestProperty

public void setRequestProperty(String key, String value)

Sets the general request property.

Parameters:

key- the keyword by which the request is known (eg "accept")

value- the value associated with it

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.setUseCaches

public void setUseCaches (boolean usecaches)

Sets the value of the useCaches (I-§4.9.7) field of this URL-Connection to the specified value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.toString

public String toString()

Returns:

a string representation of this URLConnection.

Overrides:

toString in class Object ([1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.10 Class URLEncoder

```
public class java.net.URLEncoder
    extends java.lang.Object (l-§1.12)
{
    // Methods
    public static String encode(String s); §4.10.1
}
```

The class contains a utility method for converting a String into a MIME format called "x-www-form-urlencoded" format.

To convert a String, each character is examined in turn:

- The ASCII characters 'a' through 'z', 'A' through 'Z', and '0' through '9' remain the same.
- The space character ' ' is converted into a plus sign '+'.
- All other characters are converted into the three-character string "%xy" where xy is the two-digit hexadecimal representation of the lower 16-bits of the character.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLEncoder.encode

public static String encode(String s)

Translates a String into x-www-form-urlencoded format.

Parameters:

s- String to be translated

Returns:

the translated String.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§4.11 Class URLStreamHandler

```
public abstract class java.net.URLStreamHandler
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public URLStreamHandler(); §4.11.1

    // Methods
    protected abstract URLConnection openConnection(URL u); §4.11.2
    protected void parseURL(URL u, String spec, §4.11.3
                           int start, int limit);
    protected void setURL(URL u, String protocol, §4.11.4
                          String host, int port, String file, String ref);
    protected String toExternalForm(URL u); §4.11.5
}
```

The abstract class `URLStreamHandler` is the common superclass for all stream protocol handlers. A stream protocol handler knows how to make a connection for a particular protocol type, such as http, ftp, or gopher.

In most cases, an instance of a `URLStreamHandler` subclass is not created directly by an application. Rather the first time a protocol name is encountered when constructing a URL, the appropriate stream protocol handler is automatically loaded. See I-§4.8.2 for complete details.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLStreamHandler.URLStreamHandler

public URLStreamHandler ()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLConnection.openConnection

protected abstract `URLConnection openConnection(URL u)`
throws `IOException`

Opens a connection to the object referenced by the URL argument.

Parameters:

`u` – the URL that this connects to

Returns:

a `URLConnection` object for the URL.

Throws

`IOException` (I-§2.29)

If an I/O error occurs while opening the connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLStreamHandler.parseURL

protected void

parseURL(URL u, String spec, int start, int limit)

Parses the string representation of a URL into a URL object.

If there is any inherited context then it has already been copied into the URL argument

This method parses the string representation as if it were an http specification. Most URL protocol families have a similar parsing. A stream protocol handler for a protocol that has a different syntax must override this routine.

Parameters:

u- the URL to receive the result of parsing the spec

spec- the String representing the URL which must be parsed

start- the character index at which to begin parsing; this is just past the ':' (if there is one) that specifies the determination of the protocol name.

limit- the character position to stop parsing at; this is the end of the string or the position of the '#' character, if present; all information after the sharp sign indicates an anchor

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLStreamHandler.setURL

protected void setURL(URL u, String protocol, String host, int port, String file, String ref)

Sets the fields of the URL argument to the indicated values.

Parameters:

u- the URL to modify
protocol- the protocol name
host- the remote host value for the URL
port- the port on the remote machine
file- the file
ref- the reference

{ewl msdncd.dll, ewcright, /c"Microsoft"}

URLStreamHandler.toExternalForm

protected String toExternalForm(URL u)

Converts a URL of a specific protocol to a String.

Parameters:

u- the URL

Returns:

a string representation of the URL argument.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§4.12 Interface ContentHandlerFactory

```
public interface java.net.ContentHandlerFactory
{
    // Methods
    public abstract ContentHandler §4.12.1
        createContentHandler(String mimeType);
}
```

This interface defines a factory for content handlers. An implementation of this interface should map a MIME type into an instance of ContentHandler (I-§4.1).

This interface is used by the URLStreamHandler class (I-§4.11) to create a ContentHandler for a MIME type.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ContentHandlerFactory.createContentHandler

`public abstract ContentHandler`

`createContentHandler(String mimeType)`

Creates a new ContentHandler to read an object from a URLStreamHandler.

Parameters:

`mimeType` - the MIME type for which a content handler is desired

Returns:

a new ContentHandler (I-§4.1) to read an object from a URLStreamHandler (I-§4.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.13 Interface SocketImplFactory

```
public interface java.net.SocketImplFactory
{
    // Methods
    public abstract SocketImpl createSocketImpl(); §4.13.1
}
```

This interface defines a factory for socket implementations. It is used by the classes Socket (I-§4.6) and ServerSocket (I-§4.5) to create actual socket implementations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SocketImplFactory.createSocketImpl

public abstract SocketImpl createSocketImpl()

Returns:

a new instance of SocketImpl (I-§4.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.14 Interface URLStreamHandlerFactory

```
public interface java.net.URLStreamHandlerFactory
{
    // Methods
    public abstract URLStreamHandler §4.14.1
        createURLStreamHandler(String protocol);
}
```

This interface defines a factory for URL stream protocol handlers.

It is used by the URL class (I-§4.8) to create a URLStreamHandler (I-§4.11) for a specific protocol.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


URLStreamHandlerFactory.createURLStreamHandler

`public abstract URLStreamHandler`

`createURLStreamHandler(String protocol)`

Parameters:

`protocol` - the protocol ("ftp", "http", "nntp", *and so on.*)

Returns:

a URLStreamHandler (I-§4.11) for the specific protocol.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.15 Class MalformedURLException

```
public class java.net.MalformedURLException
    extends java.io.IOException (I-§2.29)
{
    // Constructors
    public MalformedURLException(); §4.15.1
    public MalformedURLException(String msg); §4.15.2
}
```

Thrown to indicate that a malformed URL has occurred. Either no legal protocol could be found in a specification string, or the string could not be parsed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MalformedURLException.MalformedURLException

public `MalformedURLException()`

Constructs a `MalformedURLException` with no detail message.

public `MalformedURLException(String msg)`

Constructs a `MalformedURLException` with the specified detail message.

Parameters:

`msg`– the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.16 Class ProtocolException

```
public class java.net.ProtocolException
    extends java.io.IOException (l-§2.29)
{
    // Constructors
    public ProtocolException(); §4.16.1
    public ProtocolException(String host); §4.16.2
}
```

Thrown to indicate than there is an error in the underlying protocol, such as a TCP error.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ProtocolException.ProtocolException

public ProtocolException()

Constructs a new ProtocolException with no detail message.

public ProtocolException(String host)

Constructs a new ProtocolException with the specified detail message.

Parameters:

host- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.17 Class SocketException

```
public class java.net.SocketException
    extends java.io.IOException (I-§2.29)
{
    // Constructors
    public SocketException(); §4.17.1
    public SocketException(String msg); §4.17.2
}
```

Thrown to indicate that some error occurred while attempting to use a socket.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SocketException.SocketException

public SocketException()

Constructs a new SocketException with no detail message.

public SocketException(String msg)

Constructs a new SocketException with the specified detail message.

Parameters:

msg- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.18 Class UnknownHostException

```
public class java.net.UnknownHostException
    extends java.io.IOException (I-§2.29)
{
    // Constructors
    public UnknownHostException(); §4.18.1
    public UnknownHostException(String host); §4.18.2
}
```

Thrown to indicate that the IP address of a host could not be determined.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


UnknownHostException.UnknownHostException

public `UnknownHostException()`

Constructs a new `UnknownHostException` with no detail message.

public `UnknownHostException(String host)`

Constructs a new `UnknownHostException` with the specified detail message.

Parameters:

`host` - the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.19 Class UnknownServiceException

```
public class java.net.UnknownServiceException
    extends java.io.IOException (I-§2.29)
{
    // Constructors
    public UnknownServiceException(); §4.19.1
    public UnknownServiceException(String msg); §4.19.2
}
```

Thrown to indicate that an unknown service exception has occurred. Either the MIME type returned by a URL connection does not make sense, or the application is attempting to write to a read-only URL connection.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

UnknownServiceException.UnknownServiceException

public UnknownServiceException()

Constructs a new UnknownServiceException with no detail message.

public UnknownServiceException(String msg)

Constructs a new UnknownServiceException with the specified detail message.

Parameters:

msg- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.awt

Classes

- 1.1 Class BorderLayout
- 1.2 Class Button
- 1.3 Class Canvas
- 1.4 Class CardLayout
- 1.5 Class Checkbox
- 1.6 Class CheckboxGroup
- 1.7 Class CheckboxMenuItem
- 1.8 Class Choice
- 1.9 Class Color
- 1.10 Class Component
- 1.11 Class Container
- 1.12 Class Dialog
- 1.13 Class Dimension
- 1.14 Class Event
- 1.15 Class FileDialog
- 1.16 Class FlowLayout
- 1.17 Class Font
- 1.18 Class FontMetrics
- 1.19 Class Frame
- 1.20 Class Graphics
- 1.21 Class GridBagConstraints
- 1.22 Class GridBagLayout
- 1.23 Class GridLayout
- 1.24 Class Image
- 1.25 Class Insets
- 1.26 Class Label
- 1.27 Class List
- 1.28 Class MediaTracker
- 1.29 Class Menu
- 1.30 Class MenuBar
- 1.31 Class MenuComponent
- 1.32 Class MenuItem
- 1.33 Class Panel
- 1.34 Class Point
- 1.35 Class Polygon
- 1.36 Class Rectangle
- 1.37 Class Scrollbar
- 1.38 Class TextArea
- 1.39 Class TextComponent
- 1.40 Class TextField
- 1.41 Class Toolkit
- 1.42 Class Window

Interfaces

1.43 Interface LayoutManager

1.44 Interface MenuContainer

Exceptions

1.45 Class AWTException

Errors

1.46 Class AWTError

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.1 Class BorderLayout

```
public class java.awt.BorderLayout
    extends java.lang.Object (I-§1.12)
    implements java.awt.LayoutManager (II-§1.43)
{
    // Constructors
    public BorderLayout(); §1.1.1
    public BorderLayout(int hgap, int vgap); §1.1.2

    // Methods
    public void addLayoutComponent(String name, §1.1.3
        Component comp);
    public void layoutContainer(Container target); §1.1.4
    public Dimension minimumLayoutSize(Container target); §1.1.5
    public Dimension preferredLayoutSize(Container target); §1.1.6
    public void removeLayoutComponent(Component comp); §1.1.7
    public String toString(); §1.1.8
}
```

A border layout lays out a container using members named "North", "South", "East", "West", and "Center". The components get laid out according to their preferred sizes and the constraints of the container's size. The "North" and "South" components may be stretched horizontally; the "East" and "West" components may be stretched vertically; the "Center" component may stretch both horizontally and vertically to fill any space left over.

Here is an example of five buttons in an applet laid out using the BorderLayout layout manager{ewc msdncd, EWGraphic, AWT0dx 0 /a "sunref.BMP"}:

The code for this applet:

```
import java.awt.*;
import java.applet.Applet;

public class buttonDir extends Applet {
    public void init() {
        setLayout(new BorderLayout());
        add("North", new Button("North"));
        add("South", new Button("South"));
        add("East", new Button("East"));
        add("West", new Button("West"));
        add("Center", new Button("Center"));
    }
}
```



```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BorderLayout.BorderLayout

public BorderLayout()

Constructs a new border layout.

public BorderLayout(int hgap, int vgap)

Creates a new border layout with the specified horizontal and vertical gaps. The horizontal and vertical gaps specify the space between the components.

Parameters:

hgap- the horizontal gap

vgap- the vertical gap

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.addLayoutComponent

public void addLayoutComponent(String name, Component comp)

Adds the specified component to this border layout using the indicated tag.

For border layouts, the tag should be one of "North", "South", "East", "West", or "Center". Any other tag is ignored.

Most applications do not call this method directly. This method is called when a component is added to a container using the two-argument add method (II-§1.11.3).

Parameters:

name- a tag understood by the layout manager

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.layoutContainer

public void layoutContainer(Container target)

Lays out the container argument using this border layout.

This method reshapes the components in the specified container in order to satisfy the constraints of this BorderLayout object. The "North" and "South" components, if any, are placed at the top and bottom of the container, respectively. The "West" and "East" components are then placed on the left and right, respectively. Finally, the "Center" object is placed in any remaining space in the middle.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target - the container in which to do the layout

See Also:

Container (II-§1.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.minimumLayoutSize

public **Dimension** **minimumLayoutSize**(**Container** **target**)

Determines the minimum size of the **target** container using this layout manager.

This **method** is called when a container calls its **minimumSize** **method** ([II-§1.11.14](#)). Most applications do not call this **method** directly.

Parameters:

target– the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

Container ([II-§1.11](#))

preferredLayoutSize ([II-§1.1.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.preferredLayoutSize

public **Dimension** preferredLayoutSize(Container target)

Determines the preferred size of the target container using this layout manager.

This method is called when a container calls its preferredSize method [\(II-§1.11.17\)](#). Most applications do not call this method directly.

Parameters:

target– the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

Container [\(II-§1.11\)](#)

minimumLayoutSize [\(II-§1.1.5\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.removeLayoutComponent

public void removeLayoutComponent(Component comp)

Removes the specified component from this border layout. This method is called when a container calls its remove ([II-§1.11.19](#)) or removeAll ([II-§1.11.20](#)) methods. Most applications do not call this method directly.

Parameters:

comp- the component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

BorderLayout.toString

public String toString()

Returns:

a string representation of this border layout.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.2 Class Button

```
public class java.awt.Button
    extends java.awt.Component (II-§1.10)
{
    // Constructors
    public Button(); §1.2.1
    public Button(String label); §1.2.2

    // Methods
    public void addNotify(); §1.2.3
    public String getLabel(); §1.2.4
    protected String  paramString(); §1.2.5
    public void setLabel(String label); §1.2.6
}
```

This class creates a labeled button. The application can cause some action to happen when the button is pushed. Below are three views of a "Quit" button in Solaris:

{ewc msdncd, EWGraphic, AWT0dy 0 /a "sunref.BMP"}The first shows the button normally. The second shows the button when it has the input focus: the darkening of the outline lets the user know that this is an active object. The third shows the button when the user clicks the mouse over the button, and thus requests that the mouse's action(s) be performed.

When a button is pushed and released¹, AWT sends an action event (II-§1.14.11) to the button. This event's target is the button, and its object is the string label of the button. An application should override the action method (II-§1.10.1) of the button or of one of its containing windows in order to cause some action to occur.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Button.Button

public Button()

Creates a button with no label.

public Button(String label)

Creates a button with the indicated label.

Parameters:

label- a string label for the button

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Button.addNotify

public void addNotify()

This method calls the createButton method (II-§1.41.3) of the button's toolkit (II-§1.10.20) in order to create a ButtonPeer (II-§3.1) for this button. This peer allows the application to change the look of a button without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Button.getLabel

public String getLabel()

Returns:

the label of this button, or null if this button has no label.

See Also:

setLabel (II-§1.2.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Button paramString

protected String paramString()

Returns the parameter string representing the state of this button. This string is useful for debugging.

Returns:

the parameter string of this button.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Button.setLabel

```
public void setLabel(String label)
```

Changes this button's label to be the String argument.

Parameters:

label- the new label, or null for no label

See Also:

getLabel (II-§1.2.4).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Footnotes

¹In Java 1.0, the AWT does not send mouse or focus events to a button. In Java 1.1, the AWT sends the button all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.3 Class Canvas

```
public class java.awt.Canvas
    extends java.awt.Component (II-§1.10)
{
    // Constructors
    public Canvas(); §1.3.1

    // Methods
    public void addNotify(); §1.3.2
    public void paint(Graphics g); §1.3.3
}
```

A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.

An application must subclass the Canvas class in order to get useful functionality, such as creating a custom component. The paint method (II-§1.3.3) must be overridden in order to perform custom graphics on the canvas.

The AWT sends the canvas all mouse, keyboard, and focus events that occur over it. The gotFocus (II-§1.10.21), lostFocus (II-§1.10.39), keyDown (II-§1.10.31), keyUp (II-§1.10.32), mouseEnter (II-§1.10.43), mouseExit (II-§1.10.44), mouseMove (II-§1.10.45), mouseDrag (II-§1.10.42), mouseDown (II-§1.10.41), and mouseUp (II-§1.10.46) methods may be overridden in order to catch user events.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Canvas.Canvas

public Canvas ()

Creates a canvas.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Canvas.addNotify

public void addNotify()

This method calls the createCanvas method (II-§1.41.4) of the canvas's toolkit (II-§1.10.20) in order to create a CanvasPeer (II-§3.2) for this canvas. This peer allows the application to change the look of a canvas without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Canvas.paint

public void paint(Graphics g)

This method is called to repaint this canvas. Most applications that subclass Canvas should override this method in order to perform some useful operation.

The paint method provided by Canvas redraws this canvas's rectangle in the background color.

The graphics context's (0, 0) point is the top-left corner of this canvas. Its clipping region is the area of the context.

Parameters:

g- the graphics context

Overrides:

paint in class Component (II-§1.10.49).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

§1.4 Class CardLayout

```
public class java.awt.CardLayout
    extends java.lang.Object (I-§1.12)
    implements java.awt.LayoutManager (II-§1.43)
{
    // Constructors
    public CardLayout(); §1.4.1
    public CardLayout(int hgap, int vgap); §1.4.2

    // Methods
    public void addLayoutComponent(String name, Component comp); §1.4.3
    public void first(Container target); §1.4.4
    public void last(Container target); §1.4.5
    public void layoutContainer(Container target); §1.4.6
    public Dimension minimumLayoutSize(Container target); §1.4.7
    public void next(Container target); §1.4.8
    public Dimension preferredLayoutSize(Container target); §1.4.9
    public void previous(Container target); §1.4.10
    public void removeLayoutComponent(Component comp); §1.4.11
    public void show(Container target, String name); §1.4.12
    public String toString(); §1.4.13
}
```

A layout manager for a container that contains several "cards". Only one card is visible at a time, allowing the application to flip through the cards.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CardLayout.CardLayout

public CardLayout()

Creates a new card layout.

public CardLayout(int hgap, int vgap)

Creates a new card layout with the specified horizontal and vertical gaps. The horizontal gaps are placed at the left and right edge. The vertical gaps are placed at the top and bottom edge.

Parameters:

hgap- the horizontal gap

vgap- the vertical gap

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.addLayoutComponent

public void addLayoutComponent(String name, Component comp)

Adds the specified component to the card layout using the indicated name tag. The show method (II-§1.4.12) can be used to display the component with the specified tag.

Parameters:

name- a tag understood by the layout manager

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.first

public void first(Container target)

Flips to the first card of the container.

Parameters:

target- the container in which to do the layout

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CardLayout.last

public void last(Container target)

Flips to the last card of the container.

Parameters:

target- the container in which to do the layout

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CardLayout.layoutContainer

public void layoutContainer(Container target)

Lays out the container argument using this card layout.

Each component in the target container is reshaped to be the size of the container minus space for surrounding insets, horizontal gaps, and vertical gaps.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target– the container in which to do the layout

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.minimumLayoutSize

public Dimension minimumLayoutSize(Container target)

Determines the minimum size of the container argument using this card layout.

The minimum width of a card layout is the largest minimum width of the cards in the container, plus twice the horizontal gap, plus the left and right insets.

The minimum height of a card layout is the largest minimum height of the cards in the container, plus twice the vertical gap, plus the top and bottom insets.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target– the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

preferredLayoutSize (II-§1.4.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.next

public void next(Container target)

Flips to the next card of the specified container. If the currently visible card is the last one, this method flips to the first card in the layout.

Parameters:

target- the container in which to do the layout.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.preferredLayoutSize

public Dimension preferredLayoutSize(Container target)

Determines the preferred size of the container argument using this card layout.

The preferred width of a card layout is the largest preferred width of the cards in the container, plus twice the horizontal gap, plus the left and right insets.

The preferred height of a card layout is the largest preferred height of the cards in the container, plus twice the vertical gap, plus the top and bottom insets.

Most applications do not call this method directly. This method is called when a container calls its preferredSize method (II-§1.11.17).

Parameters:

target– the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

minimumLayoutSize (II-§1.4.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.previous

public void previous(Container target)

Flips to the previous card of the specified container. If the currently visible card is the first one, this method flips to the last card in the layout.

Parameters:

target- the container in which to do the layout

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.removeLayoutComponent

public void removeLayoutComponent(Component comp)

Removes the specified component from the layout.

This method is called when a container calls its remove ([II-§1.11.19](#)) or removeAll ([II-§1.11.20](#)) methods. Most applications do not call this method directly.

Parameters:

comp- the component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.show

public void show(Container target, String name)

Flips to the component that was added to this layout (II-§1.4.3) with the specified name tag.
If no such component exists, then nothing happens.

Parameters:

target- the container in which to do the layout

name- a tag

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CardLayout.toString

public String toString()

Returns:

a String representation of this card layout.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.5 Class Checkbox

```
public class java.awt.Checkbox
    extends java.awt.Component (II-§1.10)
{
    // Constructors
    public Checkbox(); §1.5.1
    public Checkbox(String label); §1.5.2
    public Checkbox(String label, CheckboxGroup group, §1.5.3
                    boolean state);

    // Methods
    public void addNotify(); §1.5.4
    public CheckboxGroup getCheckboxGroup(); §1.5.5
    public String getLabel(); §1.5.6
    public boolean getState(); §1.5.7
    protected String  paramString(); §1.5.8
    public void setCheckboxGroup(CheckboxGroup g); §1.5.9
    public void setLabel(String label); §1.5.10
    public void setState(boolean state); §1.5.11
}
```

A check box is a graphical component that has an "on" (true) and "off" (false) state. Clicking on the check box changes its state from "on" to "off" or from "off" to "on".

For example, the code:

```
setLayout(new GridLayout(3, 1));
add(new Checkbox("one", null, true));
add(new Checkbox("two"));
add(new Checkbox("three"));
```

produces the following three check boxes:

{ewc msdn cd, EWGraphic, AWT0eb 0 /a "sunref.BMP"}The button labelled one is "on". The other two are "off".

When the check box is clicked,¹AWT sends an action event (II-§1.14.11) to the check box. This event's target is the check box, and its object is a Boolean (I-§1.1) giving the new state of the check box. An application should override the action method (II-§1.10.1) of the check box or of one of its containing windows in order to cause some action to occur.

Optionally, several check boxes can be grouped together into a **CheckboxGroup** (II-§1.6). At most one button in a group can be in the "on" state at any given time. Pushing a check box to turn it "on" forces any other check box in the group that is "on" to become "off".


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Checkbox.Checkbox

public **Checkbox** ()

Creates a check box with no label. The check box is set to "off" and is not part of any check box group.

public **Checkbox** (String label)

Creates a check box with the specified label. The check box is set to "off" and is not part of any check box group.

Parameters:

label- a string label for the check box, or null for no label.

public **Checkbox** (String label, CheckboxGroup group,
boolean state)

Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Parameters:

label- a string label for the check box, or null for no label

group- this check box's check box group, or null for no group

state- the initial state of the check box; true indicates "on"; false indicates "off"

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.addNotify

public void addNotify()

This method calls the createCheckbox method (II-§1.41.5) of the check box's toolkit (II-§1.10.20) in order to create a CheckboxPeer (II-§3.4) for this button. This peer allows the application to change the look of a check box without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.getCheckboxGroup

public **CheckboxGroup** **getCheckboxGroup()**

Determines this check box's group.

Returns:

this check box's group, or null if it is not part of a check box group.

See Also:

setCheckboxGroup ([II-§1.5.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.getLabel

public String getLabel()

Returns:

the label of this check box, or null if this check box has no label.

See Also:

setLabel (II-§1.5.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.getState

public boolean getState()

Determines if this check box is "on" or "off."

Returns:

the state of this check box. The value true indicates "on", false indicates "off".

See Also:

setState [\(II-§1.5.11\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox paramString

protected String paramString()

Returns the parameter string representing the state of this check box. This string is useful for debugging.

Returns:

the parameter string of this check box.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.setCheckboxGroup

public void setCheckboxGroup (CheckboxGroup g)

Sets the group of this check box to be the specified check box group. If this check box is already in a different check box group, it is first taken out of that group.

Parameters:

g– the new check box group, or null to remove the check box from any check box group

See Also:

getCheckboxGroup [\(II-§1.5.5\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Checkbox.setLabel

```
public void setLabel(String label)
```

Sets this check box's label to be the string argument.

Parameters:

label- the new label, or null for no label

See Also:

getLabel (II-§1.5.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Checkbox.setState

public void setState(boolean state)

Sets this check box to the specified boolean state; true indicates "on"; false indicates "off."

Parameters:

state- the boolean state of the check box.

See Also:

getState ([II-§1.5.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse or focus events to a check box. In Java 1.1, the AWT sends the check box all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.6 Class CheckboxGroup

```
public class java.awt.CheckboxGroup
    extends java.lang.Object (I-§1.12)
{
    // Constructors
    public CheckboxGroup(); §1.6.1

    // Methods
    public Checkbox getCurrent(); §1.6.2
    public void setCurrent(Checkbox box); §1.6.3
    public String toString(); §1.6.4
}
```

This class is used to group together a set of check box buttons.

Exactly one check box button in a **CheckboxGroup** can be in the "on" state at any given time. Pushing any button turns it "on" and forces any other button that is "on" to become "off."

For example, the code:

```
setLayout(new GridLayout(3, 1));
CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox("one", cbg, true));
add(new Checkbox("two", cbg, false));
add(new Checkbox("three", cbg, false));
```

produces the following three check boxes:

{ewc msdncd, EWGraphic, AWT0ec 0 /a "sunref.BMP"}When a check box in a check box group is clicked, AWT sends an action event (II-§1.14.11) to that check box. This event's target is the check box, and its object is a value equal to **Boolean.TRUE** (I-§1.1.2). No action event is sent to the check box (if any) that is turned off. An application should override the action method (II-§1.10.1) of the check box or of one of its parent containers in order to cause some action to occur.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxGroup.CheckboxGroup

public **CheckboxGroup** ()

Creates a new CheckboxGroup.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxGroup.getCurrent

public `Checkbox` `getCurrent()`

Returns:

the check box in this check box group that is currently "on", or null if all are "off."

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxGroup.setCurrent

public void setCurrent(Checkbox box)

If the indicated check box argument belongs to this check box group, this method sets that check box to be "on", and all other checkboxes in this group to be "off."

If the check box argument is null or belongs to a different check box group, then this method does nothing.

Parameters:

box* the check * box to set "on"

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxGroup.toString

public String toString()

Returns:

a string representation of this check box group.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.7 Class CheckboxMenuItem

```
public class java.awt.CheckboxMenuItem
    extends java.awt.MenuItem (II-§1.32)
{
    // Constructors
    public CheckboxMenuItem(String label); §1.7.1

    // Methods
    public void addNotify(); §1.7.2
    public boolean getState(); §1.7.3
    public String  paramString(); §1.7.4
    public void setState(boolean t); §1.7.5
}
```

This class represents a check box that can be included in a menu. Clicking on the check box in the menu changes its state from "on" to "off" or from "off" to "on."

The picture of a menu bar (page II-188) shows five menu items. The item labeled Check shows a check box menu item in its "off" state.

When a check box menu item is clicked, the AWT sends an action event (II-§1.14.11) to the check box menu item's containing frame. The event's target is the check box menu item, and its object is the string label of the check box.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CheckboxMenuItem.CheckboxMenuItem

public ~~CheckboxMenuItem~~(String label)

Creates a check box with the specified label. The check box is initially set to "off."

Parameters:

label- a string label for the check box menu item, or null for an unlabeled menu item

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxMenuItem.addNotify

public void addNotify()

This method calls the createCheckboxMenuItem method (II-§1.41.6) of the check box's toolkit (II-§1.10.20) in order to create a CheckboxMenuItemPeer (II-§3.3) for this button. This peer allows the application to change the look of a check box menu item without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class MenuItem (II-§1.32.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxMenuItem.getState

public boolean getState()

Determines whether this check box menu item is "on" or "off."

Returns:

the state of this check box menu item: true indicates "on", false indicates "off."

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxMenuItem.paramString

public String paramString()

Returns the parameter string representing the state of this check box menu item. This string is useful for debugging.

Returns:

the parameter string of this check box menu item.

Overrides:

paramString in class MenuItem [\(II-§1.32.8\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxMenuItem.setState

public void setState(boolean state)

Sets this check box menu item to the specified boolean state: true indicates "on"; false indicates "off."

Parameters:

state- the boolean state of this check box menu item

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.8 Class Choice

```
public class java.awt.Choice
    extends java.awt.Component (II-§1.10)
{
    // Constructors
    public Choice(); §1.8.1

    // Methods
    public void addItem(String item); §1.8.2
    public void addNotify(); §1.8.3
    public int countItems(); §1.8.4
    public String getItem(int index); §1.8.5
    public int getSelectedIndex(); §1.8.6
    public String getSelectedItem(); §1.8.7
    protected String  paramString(); §1.8.8
    public void select(int pos); §1.8.9
    public void select(String str); §1.8.10
}
```

The Choice class presents a pop-up menu of choices. The current choice is displayed as the title of the menu.

For example, the code:

```
Choice ColorChooser = new Choice();
ColorChooser.addItem("Green");
ColorChooser.addItem("Red");
ColorChooser.addItem("Blue");
```

produces the following pop-up menu, after it has been added to a panel:

{ewc msdncd, EWGraphic, AWT0ee 0 /a "sunref.BMP"}In the picture, "Green" is the current choice. Pushing the mouse button down causes a menu to appear with the current choice highlighted.

After any choice is made,¹AWT sends an action event (II-§1.14.11) to the choice menu. The event's target is the choice menu, and its object is the string label of the currently selected item. An application should override the action method (II-§1.10.1) of the choice menu or of one of its parent containers in order to cause some action to occur.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.Choice

public Choice()

Creates a new choice menu. The menu initially has no items in it.

By default, the first item added to the choice menu becomes the selected item, until a different selection is made by the user or by calling one of the select methods ([§1.8.9](#), [§1.8.10](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Choice.addItem

public void addItem(String item)

Adds an item to this choice menu.

Parameters:

item- the item to be added

Throws

NullPointerException (I-§1.40)

If the item's value is equal to null.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.addNotify

public void addNotify()

This method calls the createChoice method (II-§1.41.7) of this object's toolkit (II-§1.10.20) in order to create a ChoicePeer (II-§3.5) for this button. This peer allows the application to change the look of a choice mneu without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.countItems

public int countItems()

Returns:

the number of menu items in this choice menu.

See Also:

getItem [\(II-§1.8.5\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.getItem

public String getItem(int index)

Parameters:

index- the index

Returns:

the string at the specified index in this choice menu

See Also:

countItems [\(II-§1.8.4\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.getSelectedIndex

public int `getSelectedIndex()`

Returns:

the index of the currently selected item in this choice menu.

See Also:

`getSelectedItem` [\(II-§1.8.7\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.getSelectedItem

public String `getSelectedItem()`

Returns:

a string representation of the currently selected item in this choice menu.

See Also:

`getSelectedIndex` [\(II-§1.8.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice paramString

protected String paramString()

Returns the parameter string representing the state of this choice menu. This string is useful for debugging.

Returns:

the parameter string of this choice menu.

Overrides:

paramString in class Component [\(II-§1.10.51\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Choice.select

public void select(int pos)

Sets the selected item in this choice menu to be the item at the specified position.

Parameters:

pos- the selected item position

Throws

IllegalArgumentException [\(I-§1.32\)](#)

If the choice item position is invalid.

See Also:

getSelectedItem [\(II-§1.8.7\)](#)

getSelectedIndex [\(II-§1.8.6\)](#).

public void select(String str)

Sets the selected item in this choice menu to be the choice whose name is equal [\(I-§1.16.14\)](#) to the specified string. If more than one choice is equal to the specified string, the one with the smallest index whose name matches is selected.

Parameters:

str- the string to select

See Also:

getSelectedItem [\(II-§1.8.7\)](#)

getSelectedIndex [\(II-§1.8.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse or focus events to a choice menu. In Java 1.1, the AWT sends the cheoice menu all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.9 Class Color

```
public final class java.awt.Color
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static Color black; §1.9.1
    public final static Color blue; §1.9.2
    public final static Color cyan; §1.9.3
    public final static Color darkGray; §1.9.4
    public final static Color gray; §1.9.5
    public final static Color green; §1.9.6
    public final static Color lightGray; §1.9.7
    public final static Color magenta; §1.9.8
    public final static Color orange; §1.9.9
    public final static Color pink; §1.9.10
    public final static Color red; §1.9.11
    public final static Color white; §1.9.12
    public final static Color yellow; §1.9.13

    // Constructors
    public Color(float r, float g, float b); §1.9.14
    public Color(int rgb); §1.9.15
    public Color(int r, int g, int b); §1.9.16

    // Methods
    public Color brighter(); §1.9.17
    public Color darker(); §1.9.18
    public boolean equals(Object obj); §1.9.19
    public int getBlue(); §1.9.20
    public static Color getColor(String nm); §1.9.21
    public static Color getColor(String nm, Color v); §1.9.22
    public static Color getColor(String nm, int v); §1.9.23
    public int getGreen(); §1.9.24
    public static Color §1.9.25
        getHSBColor(float h, float s, float b);
    public int getRed(); §1.9.26
    public int getRGB(); §1.9.27
    public int hashCode(); §1.9.28
    public static int HSBtoRGB(float hue, float saturation, §1.9.29
        float brightness);
    public static float[] RGBtoHSB(int r, int g, int b, §1.9.30
        float hsbvals[]);
    public String toString(); §1.9.31
}
```


This class encapsulates colors using the RGB format. In this format, the red, blue, and green components of a color are each represented by an integer in the range 0-255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.black

```
public final static Color black    = new Color(0, 0, 0)
```

The color black.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.blue

```
public final static Color blue = new Color(0, 0, 255)
```

The color blue.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.cyan

```
public final static Color cyan = new Color(0, 255, 255)
```

The color cyan.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.darkGray

```
public final static Color darkGray = new Color(64, 64, 64)
```

The color dark gray.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.gray

```
public final static Color gray = new Color(128, 128, 128)
```

The color gray.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.green

```
public final static Color green = new Color(0, 255, 0);
```

The color green.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.lightGray

```
public final static Color lightGray = new Color(192, 192, 192)
```

The color light gray.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.magenta

```
public final static Color magenta = new Color(255, 0, 255)
```

The color magneta.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.orange

```
public final static Color orange = new Color(255, 200, 0)
```

The color orange.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.pink

```
public final static Color pink = new Color(255, 175, 175)
```

The color pink.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.red

```
public final static Color red = new Color(255, 0, 0)
```

The color red.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.white

```
public final static Color white = new Color(255, 255, 255)
```

The color white.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.yellow

```
public final static Color yellow = new Color(255, 255, 0)
```

The color yellow.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Color.Color

public Color(float r, float g, float b)

Creates a color with the specified red, green, and blue values, where each of the values is in the range 0.0-1.0. The value 0.0 indicates no contribution from the primary color component. The value 1.0 indicates the maximum intensity of the primary color component.

The actual color used in rendering depends on finding the best match given the color space available for a given output device.

Parameters:

- r- the red component
- g- the green component
- b- the blue component

public Color(int rgb)

Creates a color with the specified RGB value, where the red component is in bits 16-23 of the argument, the green component is in bits 8-15, of the argument, and the blue component is in bits 0-7. The value 0 indicates no contribution from the primary color component.

The actual color used in rendering depends on finding the best match given the color space available for a given output device.

Parameters:

- rgb- an integer giving the red, green, and blue components

See Also:

getRGBdefault in class ColorModel ([II-§2.1.9](#)).

public Color(int r, int g, int b)

Creates a color with the specified red, green, and blue components. The three arguments must each be in the range 0-255.

The actual color used in rendering depends on finding the best match given the color space available for a given output device.

Parameters:

r- the red component

g- the green component

b- the blue component

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.brighter

public Color **brighter**()

Returns:

a brighter version of this color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.darker

public Color darker()

Returns:

a darker version of this color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.equals

public **boolean** **equals**(**Object** obj)

The result is true if the argument is not null and is a Color object that has the same red, green, and blue value as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getBlue

public int getBlue()

Returns the blue component of this color. The result is in the range 0 to 255.

Returns:

the blue component of this color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getColor

public static Color getColor(String nm)

Finds a color in the system properties.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as an integer value (see Integer.getInteger (I-§1.8.10) for information on how the string value is interpreted as an integer). This integer is then converted to a color by using the Color constructor that takes one integer argument (II-§1.9.15).

If the specified property is not found, or could not be parsed as an integer, then null is returned.

Parameters:

nm– the property name

Returns:

the color value of the property.

public static Color getColor(String nm, Color v)

Finds a color in the system properties.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as an integer value (see Integer.getInteger (I-§1.8.10) for information on how the string value is interpreted as an integer). This integer is then converted to a color by using the Color constructor that takes one integer argument (II-§1.9.15).

If the specified property is not found, or could not be parsed as an integer, then the color specified by the second argument is returned instead.

Parameters:

nm– the property name

v– default Color value

Returns:

the Color value of the property.

public static Color getColor(String nm, int v)

Finds a color in the system properties.

The first argument is treated as the name of a system property to be obtained as if by the method System.getProperty (I-§1.18.10). The string value of this property is then interpreted as an integer value (see Integer.getInteger (I-§1.8.10) for information on how the string value is interpreted as an integer).

If the specified property is not found, or could not be parsed as an integer, then the integer value v is used instead.

This integer is then converted to a color by using the Color constructor that takes one integer argument (II-§1.9.15).

Parameters:

nm- the property name

v- the default color value

Returns:

the new color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getGreen

public int getGreen()

Returns the green component of this color. The result is in the range 0 to 255.

Returns:

the green component of this color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getHSBColor

public static Color getHSBColor(float h, float s, float b)

Determines the hue, saturation, and brightness of a color. The three components should each be a floating point number in the range 0.0 is less than or equal to h, s, b is less than or equal to 1.0.

Parameters:

- h- the hue component
- s- the saturation of the color
- b- the brightness of the color

Returns:

the color object with the specified hue, saturation, and brightness.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getRed

public int getRed()

Returns the red component of this color. The result is in the range 0 to 255.

Returns:

the red component of this color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.getRGB

public int getRGB()

Calculates a single integer representing the red, green, and blue components of this color. The red, green, and blue components of the color are each scaled to be a value between 0 (absence of the color) and 255 (complete saturation). The integer returned is the number between 0 and 0xFFFFFFFF such that bits 16-23 are the red value, bits 8-15 are the green value, and bits 0-7 are the blue value.

Returns:

an integer representing this color.

See Also:

getRGBdefault in class ColorModel [\(II-§2.1.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.HSBtoRGB

`public static int`

`HSBtoRGB(float hue, float saturation, float brightness)`

Converts a color specified by hue, saturation, and brightness, to a corresponding RGB value.

Parameters:

`hue`- the hue component of the color

`saturation`- the saturation of the color

`brightness`- the brightness of the color

Returns:

the RGB value (II-§1.9.27) of the color with the indicated hue, saturation, and brightness.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.RGBtoHSB

public static float[]

RGBtoHSB(int r, int g, int b, float hsbvals[])

Converts a color specified by its red, green, and blue components to hue, saturation, and brightness.

If the hsbvals argument is null, then a new array is allocated to return the result. Otherwise, hsbvals is returned as the result, with the values put into that array.

Parameters:

r- the red component of the color

g- the green component of the color

b- the blue component of the color

hsbvals- the array to be used to return the 3 HSB values, or null

Returns:

an array of three elements containing the hue, saturation, and brightness (in that order), of the color with the indicated red, green, and blue components.

See Also:

getRGBdefault in class ColorModel ([II-§2.1.9](#))

getRGB ([II-§1.9.27](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Color.toString

public String toString()

Returns:

a string representation of this color.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.10 Class Component

```
public abstract class java.awt.Component
    extends java.lang.Object (I-§1.12)
    implements java.awt.image.ImageObserver (II-§2.11)
{
    // Methods
    public boolean action(Event evt, Object what); §1.10.1
    public void addNotify(); §1.10.2
    public Rectangle bounds(); §1.10.3
    public int checkImage(Image image, §1.10.4
                        ImageObserver observer);
    public int checkImage(Image image, §1.10.5
                        int width, int height,
                        ImageObserver observer);
    public Image createImage(ImageProducer producer); §1.10.6
    public Image createImage(int width, int height); §1.10.7
    public void deliverEvent(Event evt); §1.10.8
    public void disable(); §1.10.9
    public void enable(); §1.10.10
    public void enable(boolean cond); §1.10.11
    public Color getBackground(); §1.10.12
    public ColorModel getColorModel(); §1.10.13
    public Font getFont(); §1.10.14
    public FontMetrics getFontMetrics(Font font); §1.10.15
    public Color getForeground(); §1.10.16
    public Graphics getGraphics(); §1.10.17
    public Container getParent(); §1.10.18
    public ComponentPeer getPeer(); §1.10.19
    public Toolkit getToolkit(); §1.10.20
    public boolean gotFocus(Event evt, Object what); §1.10.21
    public boolean handleEvent(Event evt); §1.10.22
    public void hide(); §1.10.23
    public boolean imageUpdate(Image img, int flags, int x, §1.10.24
                        int y, int w, int h);
    public boolean inside(int x, int y); §1.10.25
    public void invalidate(); §1.10.26
    public boolean isEnabled(); §1.10.27
    public boolean isShowing(); §1.10.28
    public boolean isValid(); §1.10.29
    public boolean isVisible(); §1.10.30
    public boolean keyDown(Event evt, int key); §1.10.31
    public boolean keyUp(Event evt, int key); §1.10.32
    public void layout(); §1.10.33
    public void list(); §1.10.34
    public void list(PrintStream out); §1.10.35
```


public void **list**(PrintStream out, int indent); §1.10.36
public Component **locate**(int x, int y); §1.10.37
public Point **location**(); §1.10.38
public boolean **lostFocus**(Event evt, Object what); §1.10.39
public Dimension **minimumSize**(); §1.10.40
public boolean **mouseDown**(Event evt, int x, int y); §1.10.41
public boolean **mouseDrag**(Event evt, int x, int y); §1.10.42
public boolean **mouseEnter**(Event evt, int x, int y); §1.10.43
public boolean **mouseExit**(Event evt, int x, int y); §1.10.44
public boolean **mouseMove**(Event evt, int x, int y); §1.10.45
public boolean **mouseUp**(Event evt, int x, int y); §1.10.46
public void **move**(int x, int y); §1.10.47
public void **nextFocus**(); §1.10.48
public void **paint**(Graphics g); §1.10.49
public void **paintAll**(Graphics g); §1.10.50
protected String **paramString**(); §1.10.51
public boolean **postEvent**(Event evt); §1.10.52
public Dimension **preferredSize**(); §1.10.53
public boolean §1.10.54
prepareImage(Image image, ImageObserver observer);
public §1.10.55
prepareImage(Image image, int width, int height,
ImageObserver observer);
public void **print**(Graphics g); §1.10.56
public void **printAll**(Graphics g); §1.10.57
public void **removeNotify**(); §1.10.58
public void **repaint**(); §1.10.59
public void **repaint**(int x, int y, int width, int height); §1.10.60
public void **repaint**(long tm); §1.10.61
public void **repaint**(long tm, int x, int y, §1.10.62
int width, int height);
public void **requestFocus**(); §1.10.63
public void **reshape**(int x, int y, int width, int height); §1.10.64
public void **resize**(Dimension d); §1.10.65
public void **resize**(int width, int height); §1.10.66
public void **setBackground**(Color c); §1.10.67
public void **setFont**(Font f); §1.10.68
public void **setForeground**(Color c); §1.10.69
public void **show**(); §1.10.70
public void **show**(boolean cond); §1.10.71
public Dimension **size**(); §1.10.72
public String **toString**(); §1.10.73
public void **update**(Graphics g); §1.10.74
public void **validate**(); §1.10.75

}

The Component class is the abstract superclass of many of the Abstract Window Toolkit classes. It represents something that has a position, a size, can be painted on the screen, and can receive input events.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Component.action

public boolean action(Event evt, Object what)

This method is called when an action occurs inside this component. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the `what` argument contains the `arg` field of the event argument. The specific value and type of the `what` argument depends on the component that originally triggered the action.

The method returns `true` to indicate that it successfully handled the action or `false` if the event that triggered the action should be passed up to the component's parent. Most applications should return either `true` or the value of `super.handleEvent(evt)`.

The action method of `Component` simply returns `false`.

Parameters:

`evt` - the event that caused the action

`what` - the action

Returns:

`true` if the event has been handled and no further action is necessary; `false` if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.addNotify

public void addNotify()

This method notifies the Component to create a peer. This peer allows the application to change the look of a component without changing its functionality.

The addNotify method of Component sets a flag indicating that the component needs to be laid out again because its size has possibly changed.

Most applications do not call this method directly.

See Also:

getPeer [\(II-§1.10.19\)](#)

removeNotify [\(II-§1.10.58\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.bounds

public **Rectangle** bounds()

Returns:

the containing rectangle of this component.

See Also:

reshape [\(II-§1.10.64\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.checkImage

```
public int checkImage(Image image, ImageObserver observer)
```

Returns the status of the construction of a screen representation of the specified image.

This method does not cause the image to begin loading. An application must use the prepareImage (II-§1.10.55) method to force the loading of an image.

The checkImage method of Component calls its peer's checkImage method (II-§3.6.1) to calculate the flags. If this component does not yet have a peer, the component's toolkit's checkImage method (II-§1.41.2) is called instead.

Information on the flags returned by this method can be found in II-§2.11.

Parameters:

image– the Image whose status is being checked

observer– the ImageObserver object to be notified as the image is being prepared

Returns:

the bitwise inclusive OR of ImageObserver (II-§2.11) flags indicating what information about the image is currently available.

```
public int checkImage(Image image, int width, int height, ImageObserver  
observer)
```

Returns the status of the construction of a scaled screen representation of the specified image.

This method does not cause the image to begin loading. An application must use the prepareImage (II-§1.10.55) method to force the loading of an image.

The checkImage method of Component calls its peer's checkImage method (II-§3.6.1) to calculate the flags. If this component does not yet have a peer, the component's toolkit's checkImage method (II-§1.41.2) is called instead.

Information on the flags returned by this method can be found in [II-§2.11](#).

Parameters:

`image`- the Image whose status is being checked

`width`- the width of the scaled version to check the status of

`height`- the height of the scaled version to check the status of

`observer`- the ImageObserver object to be notified as the image is being prepared

Returns:

the bitwise inclusive OR of the ImageObserver flags for the data that is currently available.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.createImage

public Image createImage(ImageProducer producer)

Creates an image from the specified image producer.

Parameters:

producer- the image producer

Returns:

the image produced.

public Image createImage(int width, int height)

Parameters:

width- the specified width

height- the specified height

Returns:

an off-screen drawable image, which can be used for double buffering.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.deliverEvent

public void deliverEvent(Event evt)

Delivers an event to this component or one of its subcomponents.

The deliverEvent method of Component calls the component's postEvent method (II-§1.10.52) on the event.

Parameters:

evt- the event

See Also:

handleEvent (II-§1.10.22).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.disable

public void disable()

Makes this component insensitive to user input.

See Also:

isEnabled ([II-§1.10.27](#))

enable ([II-§1.10.10](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.enable

public void enable()

Makes this component sensitive to user input. This is the default.

See Also:

isEnabled ([II-§1.10.27](#))

disable ([II-§1.10.9](#)).

public void enable(boolean cond)

If the boolean argument is true, enables this component; if false, disables it.

Parameters:

cond- a boolean indicating whether to enable or disable this component

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.GetBackground

public Color GetBackground()

Determines the background color of this component. If this component has not specified a background color using the SetBackground method (II-§1.10.67), the background color of its parent component is returned.

Returns:

the background color of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getColorModel

public **ColorModel** **getColorModel**()

Determines the color model of this component. The **ColorModel** [\(II-§2.1\)](#) is an abstract class that encapsulates how to translate between pixel values of an image and its red, green, blue, and alpha (transparency) components.

The **getColorModel** method of **Component** calls its peer's **getColorModel** method [\(II-§3.6.7\)](#) to determine the component's color model. If this component does not yet have a peer, the component's toolkit's **getColorModel** method [\(II-§1.41.22\)](#) is called instead.

Returns:

the color model of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getFont

public **Font** **getFont()**

Determines the font of this component. If this component has not yet specified a font using the **setFont** method (II-§1.10.68), the font of its parent component is returned.

Returns:

the font of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getFontMetrics

public **FontMetrics** **getFontMetrics**(**Font** font)

Determines the font metrics for the specified font when rendered by a platform-dependent toolkit.

The `getFontMetrics` method checks to see if this component has a peer (II-§1.10.19). If so, the peer's `getFontMetrics` method (II-§3.6.8) is called. Otherwise, the component's toolkit's `get+FontMetrics` method (II-§1.41.25) is called.

Parameters:

font- the font

Returns:

the font metrics for the specified font.

See Also:

`getFont` (II-§1.10.14).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.setForeground

public Color getForeground()

Determines the foreground color of this component. If this component has not specified a background color using the setForeground method (II-§1.10.69), the background color of its parent component is returned.

Returns:

the foreground color of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getGraphics

public Graphics getGraphics()

Returns:

the graphics context of this component; this method returns null if the component does not currently have a peer.

See Also:

paint (II-§1.10.49).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getParent

public Container getParent()

Returns:

the parent of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getPeer

public ComponentPeer getPeer()

Returns:

the peer of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.getToolkit

`public Toolkit getToolkit()`

The toolkit is used to create this component's peer when its `addNotify` method ([II-§1.10.2](#)) is called.

A Component's toolkit is determined by its containing Frame ([II-§1.19](#)).

Returns:

the toolkit of this component.

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

Component.getFocus

`public boolean getFocus(Event evt, Object what)`

This method is called when the this component receives the input focus. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the what argument contains the `arg` field of the event argument.

The method returns `true` to indicate that it has successfully handled the action, or `false` if the event that triggered the action should be passed up to the component's parent. Most applications should return either `true` or the value of `super.handleEvent(evt)`.

The `getFocus` method of `Component` simply returns `false`.

The what argument is currently always `null`.

Parameters:

`evt`– the event that caused the action

`what`– the action

Returns:

`true` if the event has been handled and no further action is necessary; `false` if the event is to be given to the component's parent.

See Also:

`requestFocus` ([II-§1.10.63](#))

`lostFocus` ([II-§1.10.39](#)).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

Component.handleEvent

`public boolean handleEvent(Event evt)`

This method is called when any event occurs inside this component.

The method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the component's parent.

The handleEvent method of Component determines the type of event, and calls one of the following methods, if appropriate:

- action ([II-§1.10.1](#))
- gotFocus ([II-§1.10.21](#))
- lostFocus ([II-§1.10.39](#))
- keyDown ([II-§1.10.31](#))
- keyUp ([II-§1.10.32](#))
- mouseEnter ([II-§1.10.43](#))
- mouseExit ([II-§1.10.44](#))
- mouseMove ([II-§1.10.45](#))
- mouseDrag ([II-§1.10.42](#))
- mouseDown ([II-§1.10.41](#))
- mouseUp ([II-§1.10.46](#))

All the above methods are called with the event as their first argument. The action method is called with the arg field of the event as its second argument. The keyUp and keyDown are passed the key field of the event as its second argument. The six mouse methods are passed the x and y fields of the event as their second and third argument, respectively.¹

Parameters:

evt- the event

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.hide

public void hide()

Hides this component. The component continues to exist and be in its containing object, but it is not visible and does not have any space allocated for it.

See Also:

isVisible [\(II-§1.10.30\)](#)

show [\(II-§1.10.70\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.imageUpdate

public boolean

imageUpdate(Image img, int flags, int x, int y, int w, int h)

This imageUpdate method of an ImageObserver (such as Component) is called when more information about an image which had been previously requested using an asynchronous interface (such as draw-Image) becomes available. See §2.11.9 for more information on this method and its arguments.

The imageUpdate method of Component incrementally draws an image on the component as more of the bits of the image are available.

If the system property "awt.image.incrementalDraw" is missing or has the value "true", the image is incrementally drawn. If the system property has any other value, then the image is not drawn until it has been completely loaded.

Also, if incremental drawing is in effect, the value of the system property "awt.image.redrawrate" is interpreted as an integer to give the maximum redraw rate, in milliseconds. If the system property is missing or cannot be interpreted as an integer, the redraw rate is once every 100ms.

The interpretation of the x, y, width, and height arguments depends on the infoflags argument.

Parameters:

img- the image being observed

infoflags- see §2.11.9 for more information

x- the x coordinate

y- the y coordinate

width- the width

height- the height

Returns:

true if the flags have indicated that the image is completely loaded; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.inside

public **boolean** **inside**(**int** **x**, **int** **y**)

Determines if the specified (x, y) location is inside this component.

The inside method of Component returns true if the (x, y) location is inside the bounding box (II-§1.10.3) of the component.

Parameters:

x- the x coordinate

y- the y coordinate

Returns:

true if the specified (x, y) location lies within this component; false otherwise.

See Also:

locate (II-§1.10.37).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.invalidate

public void invalidate()

Invalidates this component. This component is marked as having changed. The next call to the validate [\(II-§1.10.75\) method](#) on this component or its parent(s) causes the component to be laid out again.

See Also:

layout [\(II-§1.10.33\)](#)

LayoutManager [\(II-§1.43\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.isEnabled

public boolean isEnabled()

Indicates whether this component is enabled to receive events. By default, components are initially enabled.

A component is disabled by calling its disable method (II-§1.10.9). A component is enabled by calling its enable method (II-§1.10.10).

Returns:

true if this component is enabled; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.isShowing

`public boolean isShowing()`

Indicates whether this component is visible. A component is showing if it is visible ([II-§1.10.30](#)) and is inside a container this is both visible and showing.

A component is hidden by calling its [hide method \(II-§1.10.23\)](#). A component is made visible by calling its [show method \(II-§1.10.70\)](#).

Returns:

true if this component is showing; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.IsValid

public **boolean** **IsValid()**

Indicates whether this component is valid. A component is valid if it has a peer and its subcomponents have been properly laid out.

Returns:

true if this component is valid; false otherwise.

See Also:

validate [\(II-§1.10.75\)](#)

invalidate [\(II-§1.10.26\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.isVisible

public **boolean** **isVisible()**

Indicates whether this component is visible. Except for top-level components such as a frame, components are visible unless they have been specifically been made invisible by a call to the hide method (II-§1.10.23).

Returns:

true if this component is showing; false otherwise.

See Also:

show (II-§1.10.70).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.keyDown

```
public boolean keyDown(Event evt, int key)
```

This method is called when a key is pressed and this component has the focus. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the key argument contains the key field of the event argument.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The `keyDown` method of `Component` simply returns false.

Note that the key argument is an `int` rather than a `char`.

Parameters:

`evt`– the event that caused the action

`key`– the key that has been pressed

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Component.keyUp

public boolean keyUp(Event evt, int key)

This method is called when a key is released and this component has the focus. This method is usually called by `handleEvent` (II-§1.10.22), in which case the key argument contains the key field of the event argument.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The `keyUp` method of `Component` simply returns false.

Note that the key argument is an int rather than a char.

Parameters:

`evt`– the event that caused the action

`key`– the key that has been pressed

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.layout

public void layout()

This method is called to lay out the subcomponents of this component so that they fit inside the borders of this component.

This method is called when this component is validated by a call to the validate method (II-§1.10.75). Most applications should not call this method directly.

The layout method of Component does nothing.

See Also:

LayoutManager (II-§1.43).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.list

public void list()

Prints a listing of this component to System.out (I-§1.18.3).

public void list(PrintStream out)

Prints a listing of this component to the specified output stream.

Parameters:

out- a print stream

public void list(PrintStream out, int indent)

Prints a listing of this component to the specified output stream. The listing starts at the specified indentation.

The list method of Component calls the println method (I-§2.18.23) on the component after indenting by the specified amount.

Parameters:

out- a print stream

indent- number of spaces to indent

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.locate

public Component locate(int x, int y)

Determines if this component or one of its subcomponents contains the (x, y) coordinate, and if so returns the containing component. This method only looks one level deep. If the point (x, y) is inside a subcomponent that itself has subcomponents, it does not go looking down the subcomponent tree.

The locate method of Component simply returns the component itself if the (x, y) coordinate is inside (II-§1.10.25) its bounding box, and null otherwise.

Parameters:

x- the x coordinate

y- the y coordinate

Returns:

the component or subcomponent that contains the (x, y) coordinate; null if the coordinate is outside this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.location

public Point location()

Returns:

the location of this component in its parent's coordinate space.

See Also:

move [\(II-§1.10.47\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.lostFocus

public boolean lostFocus(Event evt, Object what)

This method is called when this component loses the input focus. This method is usually called by `handleEvent` :[\(II-§1.10.22\)](#), in which case the what argument contains the arg field of the event argument.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The `getFocus` method of `Component` simply returns false.

The what argument is currently always null.

Parameters:

`evt`– the event that caused the action

`what`– the action that's occurring

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

See Also:

`requestFocus` [\(II-§1.10.63\)](#)

`gotFocus` [\(II-§1.10.21\)](#).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

Component.minimumSize

public **Dimension** **minimumSize()**

Determines the minimum size of this component.

The **minimumSize** method of **Component** checks to see if the component has a peer (II-§1.10.19). If so, the **minimumSize** request is passed to the peer (II-§3.6.13). Otherwise, the minimum size is the width and height specified by the most recent reshape (II-§1.10.64) or **resize** (II-§1.10.65) method call.

Returns:

the minimum size of this component.

See Also:

preferredSize (II-§1.10.53)

LayoutManager (II-§1.43).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.MouseDown

public boolean mouseDown(Event evt, int x, int y)

This method is called when the mouse button is pushed inside this component. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the x and y arguments contains the x and y field of the event argument. The (x, y) coordinate is relative to the top-left corner of this component.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The mouseDown method of Component returns false.

Parameters:

evt- the event that caused the action

x- the x coordinate

y- the y coordinate

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.mouseDrag

public boolean mouseDrag(Event evt, int x, int y)

This method is called when the mouse button is moved inside this component with the button pushed. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the x and y arguments contain the x and y field of the event argument. The (x, y) coordinate is relative to the top-left corner of this component.

Mouse drag events continue to get sent to this component even when the mouse has left the bounds of the component. The drag events continue until a mouse up event occurs.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The mouseDrag method of Component returns false.

Parameters:

evt- the event that caused the action

x- the x coordinate

y- the y coordinate

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.mouseEnter

public boolean mouseEnter(Event evt, int x, int y)

This method is called when the mouse first enters this component. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the x and y arguments contains the x and y field of the event argument. The (x, y) coordinate is relative to the top-left corner of this component.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The `mouseEnter` method of `Component` returns false.

Parameters:

evt- the event that caused the action

x- the x coordinate

y- the y coordinate

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.mouseExit

```
public boolean mouseExit(Event evt, int x, int y)
```

This method is called when the mouse exits this component. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the `x` and `y` arguments contains the `x` and `y` field of the event argument. The `(x, y)` coordinate is relative to the top-left corner of this component. Most applications should return either `true` or the value of `super.handleEvent(evt)`.

This method returns `true` to indicate that it has successfully handled the action; or `false` if the event that triggered the action should be passed up to this component's parent. Most applications should return either `true` or the value of `super.handleEvent(evt)`.

The `mouseExit` method of `Component` returns `false`.

Parameters:

`evt`– the event that caused the action

`x`– the `x` coordinate

`y`– the `y` coordinate

Returns:

`true` if the event has been handled and no further action is necessary; `false` if the event is to be given to the component's parent.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Component.mouseMove

public boolean mouseMove(Event evt, int x, int y)

This method is called when the mouse is moved inside this component with the mouse button not pushed. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the x and y arguments contains the x and y field of the event argument. The (x, y) coordinate is relative to the top-left corner of this component.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The `mouseMove` method of `Component` returns false.

Parameters:

evt- the event that caused the action

x- the x coordinate

y- the y coordinate

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.mouseUp

public boolean mouseUp(Event evt, int x, int y)

This method is called when the mouse button is released inside this component. This method is usually called by `handleEvent` ([II-§1.10.22](#)), in which case the x and y arguments contains the x and y field of the event argument. The (x, y) coordinate is relative to the top-left corner of the component.

This method returns true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to this component's parent. Most applications should return either true or the value of `super.handleEvent(evt)`.

The mouseUp method of Component returns false.

Parameters:

evt- the event that caused the action

x- the x coordinate

y- the y coordinate

Returns:

true if the event has been handled and no further action is necessary; false if the event is to be given to the component's parent.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.move

public void move(int x, int y)

Moves this component to the coordinate (x, y) in the parent's coordinate space. The (x, y) coordinate is relative to the top-left corner of the parent component.

Components that are in a container with a layout manager should not call this method explicitly.

Parameters:

x- the x coordinate

y- the y coordinate

See Also:

location [\(II-§1.10.38\)](#)

reshape [\(II-§1.10.64\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.nextFocus

public void nextFocus()

Moves the input focus to the next component, as determined by the component's peer (II-§3.6.14).

See Also:

requestFocus (II-§1.10.63)

gotFocus (II-§1.10.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.paint

public void paint(Graphics g)

Paints this component. Most application components, including applets, override this method.

The paint method of Component calls the repaint method (II-§3.6.19) of this component's peer.

The (0, 0) coordinate of the graphics context is the top-left corner of this component. The clipping region of the graphics context is the bounding rectangle of this component.

The paint method of Component calls the repaint method (II-§3.6.19) of this component's peer.

Parameters:

g– the graphics context to use for painting

See Also:

update (II-§1.10.74).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.paintAll

public void paintAll(Graphics g)

Paints this component and all of its subcomponents.

The (0, 0) coordinate of the graphics context is the top-left corner of this component. The clipping region of the graphics context is the bounding rectangle of this component.

Parameters:

g- the graphics context to use for painting

See Also:

paint (II-§1.10.49).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.paramString

protected String paramString()

Returns the parameter string representing the state of this component. This string is useful for debugging.

Returns:

the parameter string of this component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.postEvent

public boolean postEvent(Event evt)

Posts an event to this component by calling its handleEvent method (II-§1.10.22). If handleEvent returns false, the event is posted to this component's parent.

If this component and all of its parents return false, the event is passed to this component's peer object's handleEvent method (II-§3.6.11).

Parameters:

evt- the event

Returns:

true if this component, one of its parents, or this component's peer handled the event; false otherwise.

See Also:

deliverEvent (II-§1.10.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.preferredSize

public Dimension this()

Determines the preferred size of the component.

The preferredSize method of Component checks to see if this component has a peer (II-§1.10.19). If so, the preferredSize request is passed to the peer (II-§3.6.16). Otherwise, the preferred size is the width and height specified by the most recent reshape (II-§1.10.64) or resize (II-§1.10.65) method call.

Returns:

the preferred size of this component.

See Also:

minimumSize (II-§1.10.40)

LayoutManager (II-§1.43).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.prepareImage

public boolean

prepareImage(Image image, ImageObserver observer) Prepares an image for rendering on this component.

The image data is downloaded asynchronously in another thread and the appropriate screen representation of the image is generated.

Parameters:

image- the image to prepare a screen representation for

observer- the ImageObserver ([II-§2.11](#)) object to be notified as the image is being prepared

Returns:

true if the image has already been fully prepared; false otherwise.

public boolean

prepareImage(Image image, int width, int height, ImageObserver observer)

Prepares an image for rendering on this component at the specified width and height.

The image data is downloaded asynchronously in another thread and an appropriately scaled screen representation of the image is generated.

Parameters:

image- the image to prepare a screen representation for

width- the width of the desired screen representation

height- the height of the desired screen representation

observer- the ImageObserver ([II-§2.11](#)) object to be notified as the image is being prepared

Returns:

true if the image has already been fully prepared; false otherwise.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Component.print

public void print(Graphics g)

Prints this component. Applications should override this method for components that must do special processing before being printed.

The print method of Component calls the paint method (II-§1.10.49).

The (0, 0) coordinate of the graphics context is the top-left corner of this component. The clipping region of the graphics context is the bounding rectangle of this component.

Parameters:

g– the graphics context to use for printing

See Also:

paint (II-§1.10.49).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.printAll

public void printAll(Graphics g)

Paints this component and all of its subcomponents.

The (0, 0) coordinate of the graphics context is the top-left corner of this component. The clipping region of the graphics context is the bounding rectangle of this component.

Parameters:

g- the graphics context to use for printing

See Also:

print ([II-§1.10.56](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.removeNotify

public void removeNotify()

Notifies this component and all its subcomponents to destroy their peers. Applications should not call this method directly. It is called when a component is removed (§1.11.19§1.11.20) from its container.

The removeNotify method of Component removes the components peer and has the peer clean up after itself by calling its dispose method (II-§3.6.5).

See Also:

getPeer (II-§1.10.19)

addNotify (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.repaint

public void repaint()

Repaints this component.

This method causes a call to this component's update method (II-§1.10.74) as soon as possible.

public void

repaint(int x, int y, int width, int height)

Repaints the specified rectangle of this component.

This method causes a call to this component's update method (II-§1.10.74) as soon as possible.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width

height- the height

public void repaint(long tm)

Repaints the component within tm milliseconds.

This method causes a call to this component's update method (II-§1.10.74).

Parameters:

tm- maximum time in milliseconds before update


```
public void repaint(long tm, int x, int y, width, int height)
```

Repaints the specified rectangle of this component within tm milliseconds.

This method causes a call to this component's update method (II-§1.10.74).

Parameters:

tm- maximum time in milliseconds before update

x- the x coordinate

y- the y coordinate

width- the width

height- the height

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Component.requestFocus

public void requestFocus()

Requests that this component get the input focus.

This component's gotFocus method (II-§1.10.21) is called when this method is successful.

The requestFocus method of Component calls the requestFocus method (II-§3.6.21) of the component's peer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.reshape

public void reshape(int x, int y, int width, int height)

Reshapes this component to the specified bounding box in its parent's coordinate space. Components that are in a container with a layout manager should not call this method explicitly.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of this component

height- the height of this component

See Also:

bounds ([II-§1.10.3](#))

move ([II-§1.10.47](#))

resize ([II-§1.10.66](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.resize

public void **resize**(Dimension d)

Resizes this component to the width and height specified by the dimension argument. Components that are in a container with a layout manager should not call this method explicitly.

Parameters:

d- the new dimension for this component

See Also:

size ([II-§1.10.72](#))

reshape ([II-§1.10.64](#)).

public void **resize**(int width, int height)

Resizes this component to the specified width and height. Components that are in a container with a layout manager should not call this method explicitly.

Parameters:

width- the new width of this component

height- the new height of this component

See Also:

size ([II-§1.10.72](#))

reshape ([II-§1.10.64](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.setBackground

public void setBackground(Color c)

Sets the background color of this component.

The setBackground method of Component calls the setBackground method (II-§3.6.23) of the component's peer.

Parameters:

c– the color

See Also:

getBackground (II-§1.10.12).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.setFont

public void setFont(Font f)

Sets the font of this component.

The setFont method of Component calls the setFont method (II-§3.6.24) of the component's peer.

Parameters:

f – the font

See Also:

getFont (II-§1.10.14).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.setForeground

public void setForeground(Color c)

Sets the foreground color of this component.

The setForeground method of Component calls the setForeground method (II-§3.6.25) of the component's peer.

Parameters:

c– the color

See Also:

getForeground (II-§1.10.16).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.show

public void show()

Shows this component; if this component had been made invisible by a call to the hide method [\(II-§1.10.23\)](#), makes this component visible again.

See Also:

isVisible [\(II-§1.10.30\)](#).

public void show(boolean cond)

If the boolean argument is true, makes this component visible. If false, makes this component invisible.

Parameters:

cond- if true, show this component; if false, hides this component.

See Also:

show [\(II-§1.10.70\)](#)

hide [\(II-§1.10.23\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.size

public Dimension size()

Returns:

the current size of this component.

See Also:

resize [\(II-§1.10.65\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.toString

public String toString()

Returns:

a string representation of this component.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.update

public void update(Graphics g)

Updates this component.

The AWT calls the update method in response to a call to repaint ([II-§1.10.59](#)). The appearance of the component on the screen has not changed since the last call to update or paint.

The update method of Component:

- Clears this component by filling it with the background color.
- Sets the color of the graphics context to be the foreground color of this component.
- Calls this component's paint method ([II-§1.10.49](#)) to completely redraw this component. The (0, 0) coordinate of the graphics context is the top-left corner of this component. The clipping region of the graphics context is the bounding rectangle of this component.

Parameters:

g– the graphics context to use for updating

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Component.validate

public void validate()

Validates this component if necessary. This component and any subcomponents are laid out (II-§1.10.33) again, if necessary.

See Also:

invalidate (II-§1.10.26)
LayoutManager (II-§1.43).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, AWT does not send all components mouse, keyboard, and focus events. Each component documents the events that AWT sends it.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.11 Class Container

```
public abstract class java.awt.Container
    extends java.awt.Component (II-§1.10)
{
    // Methods
    public Component add(Component comp); §1.11.1
    public Component add(Component comp, int pos); §1.11.2
    public Component add(String name, Component comp); §1.11.3
    public void addNotify(); §1.11.4
    public int countComponents(); §1.11.5
    public void deliverEvent(Event evt); §1.11.6
    public Component getComponent(int n); §1.11.7
    public Component[] getComponents(); §1.11.8
    public LayoutManager getLayout(); §1.11.9
    public Insets insets(); §1.11.10
    public void layout(); §1.11.11
    public void list(PrintStream out, int indent); §1.11.12
    public Component locate(int x, int y); §1.11.13
    public Dimension minimumSize(); §1.11.14
    public void paintComponents(Graphics g); §1.11.15
    protected String  paramString(); §1.11.16
    public Dimension preferredSize(); §1.11.17
    public void printComponents(Graphics g); §1.11.18
    public void remove(Component comp); §1.11.19
    public void removeAll(); §1.11.20
    public void removeNotify(); §1.11.21
    public void setLayout(LayoutManager mgr); §1.11.22
    public void validate(); §1.11.23
}
```

Container is the abstract superclass representing all components that can hold other components.

Each container may be associated with a LayoutManager instance (II-§1.43) that determines the position of each of the container's subcomponents.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Container.add

public Component add(Component comp)

Adds the specified component to the end of this container.

Parameters:

comp- the component to be added

Returns:

the component argument.

public Component add(Component comp, int pos)

Adds the specified component to this container at the given position.

Parameters:

comp- the component to be added

pos- the position at which to insert the component; or -1 to insert at the end

Returns:

the component argument.

See Also:

remove (II-§1.11.19).

public Component add(String name, Component comp)

Adds the specified component to the end of this container. Also adds the component to the layout manager using the name specified.

Parameters:

name- a tag understood by the layout manager

comp- the component to be added

Returns:

the component argument.

See Also:

remove [\(II-§1.11.19\)](#)
LayoutManager [\(II-§1.43\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.addNotify

public void addNotify()

Notifies this container to create a peer.

The addNotify method of Container calls the addNotify method for each of the components in this container. It then calls its superclass's addNotify method (II-§1.10.2), to indicate that the container needs to be laid out again since its size may have changed.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

See Also:

removeNotify (II-§1.11.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.countComponents

public int countComponents()

Returns:

the number of components in this container.

See Also:

GetComponent [\(II-§1.11.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.deliverEvent

public void deliverEvent(Event evt)

Delivers an event to this component or one of its subcomponents.

The deliverEvent method of Container determines whether this event properly belongs to one of its subcomponents. If so, it translates the event into the subcomponent's coordinate system and delivers the event to it by calling its deliverEvent method (II-§1.10.8).

If the event doesn't properly belong to one of the container's subcomponents, it calls this container's postEvent method (II-§1.10.52) on the event.

Parameters:

evt- the event

Overrides:

deliverEvent in class Component (II-§1.10.8).

See Also:

handleEvent in class Component (II-§1.10.22).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.getComponent

public Component getComponent(int n)

Parameters:

n- the index of the component to get

Returns:

the nth component in this container.

Throws

ArrayIndexOutOfBoundsException ([I-§1.25](#))

If the nth value does not exist.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.getComponents

public Component[] getComponents ()

Returns:

an array of all the component in this container.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.getLayout

public **LayoutManager** getLayout()

Returns:

the layout manager for this container.

See Also:

layout [\(II-§1.11.11\)](#)

setLayout [\(II-§1.11.22\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container insets

public Insets insets()

Determines the insets of this container, which indicate the size of the container's border.

A frame, for example, has a top inset that corresponds to the height of the frame's title bar.

The insets method of Container calls the insets method (II-§3.7.1) of this container's peer, if the container has a peer. Otherwise it returns the inset new Inset(0, 0, 0, 0), which indicates that the container has no border.

Returns:

the insets of this container.

See Also:

LayoutManager (II-§1.43).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.layout

public void layout()

Lays out this container.

The layout method of Container calls the layoutContainer method (II-§1.43.2) of the container's layout manager.

Most applications do not call this method directly. This method is called when a container calls its validate method (II-§1.11.23).

Overrides:

layout in class Component (II-§1.10.33).

See Also:

setLayout (II-§1.11.22).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.list

public void list(PrintStream out, int indent)

Prints a listing of this container to the specified output stream. The listing starts at the specified indentation.

The list method of Container prints itself by calling its superclass's list method (II-§1.10.36) and then calls list on each of its subcomponents with an indentation of `indent + 1`.

Parameters:

`out`— a print stream

`indent`— the number of spaces to indent

Overrides:

list in class Component (II-§1.10.36).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.locate

public Component locate(int x, int y)

Determines the component or subcomponent of this container that contains the (x, y) coordinate. This method only looks one level deep. If the point (x, y) is inside a subcomponent that itself has subcomponents, it does not look down the subcomponent tree.

The locate method of Container first determines if the (x, y) coordinate is inside (II-§1.10.25) its own boundaries. If not, it returns null immediately. If the coordinate is inside its boundaries, it calls inside (II-§1.10.25) on each of the subcomponents; if any of those calls to returns true, that subcomponent is returned; otherwise the container target object is returned.

Parameters:

x- the x coordinate

y- the y coordinate

Returns:

this container or one of its subcomponent that contains the (x, y) coordinate; null if the coordinate is outside this container.

Overrides:

locate in class Component (II-§1.10.37).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.minimumSize

public **Dimension** **minimumSize()**

Determines the minimum size of this container.

The minimumSize method of Container checks to see if this container has a layout manager (II-§1.11.9). If so, its minimumLayoutSize method (II-§1.43.3) is called. Otherwise, its superclass's minimumSize method (II-§1.10.40) is called.

Returns:

the minimum size of this container.

Overrides:

minimumSize in class Component (II-§1.10.40).

See Also:

preferredSize (II-§1.11.17).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.paintComponents

public void paintComponents(Graphics g)

Paints each of the components in this container.

Parameters:

g– the graphics context

See Also:

paint in class Component [\(II-§1.10.49\)](#)

paintAll in class Component [\(II-§1.10.50\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container paramString

protected String paramString()

Returns the parameter string representing the state of this container. This string is useful for debugging.

Returns:

the parameter string of this container.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.preferredSize

public **Dimension** preferredSize()

Determines the preferred size of this container.

The preferredSize method of Container checks to see if this container has a layout manager (II-§1.11.9). If so, its preferredLayoutSize method (II-§1.43.4) is called. Otherwise, its superclass's preferredSize method (II-§1.10.53) is called.

Returns:

the minimum size of this container.

Overrides:

preferredSize in class Component (II-§1.10.53).

See Also:

minimumSize (II-§1.11.14).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.printComponents

public void printComponents(Graphics g)

Prints each of the components in this container.

Parameters:

g– the graphics context

See Also:

print in class Component ([II-§1.10.56](#))

printAll in class Component ([II-§1.10.57](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.remove

public void remove(Component comp)

Removes the specified component from this container. This method also causes the component to call its removeNotify method (II-§1.10.58) to remove its peer.

Parameters:

comp- the component to be removed

See Also:

add (II-§1.11.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.removeAll

public void removeAll()

Removes all the components from this container. This method also causes all the components in the container to call their removeNotify method (II-§1.10.58) to remove their peers.

See Also:

add (II-§1.11.1)
remove (II-§1.11.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.removeNotify

public void removeNotify()

Notifies this container and all its subcomponents to destroy their peers.

Overrides:

removeNotify in class Component [\(II-§1.10.58\)](#).

See Also:

addNotify [\(II-§1.11.4\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Container.setLayout

public void `setLayout`(`LayoutManager mgr`)

Sets the layout manager for this container.

Parameters:

`mgr` – the new layout manager

See Also:

`layout` ([II-§1.11.11](#))

`getLayout` ([II-§1.11.9](#)).

{`ewl msdncd.dll`, `ewcright`, `/c"Microsoft"`}

Container.validate

public void validate()

Validates this container and all of its subcomponents. The AWT uses this method to have a container be laid out again after adding or otherwise changing the components it contains.

Overrides:

validate in class Component [\(II-§1.10.75\)](#).

See Also:

invalidate in class Component [\(II-§1.10.26\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.12 Class Dialog

```
public class java.awt.Dialog
    extends java.awt.Window (II-§1.42)
{
    // Constructors
    public Dialog(Frame parent, boolean modal); §1.12.1
    public Dialog(Frame parent, String title, §1.12.2
                  boolean modal);

    // Methods
    public void addNotify(); §1.12.3
    public String getTitle(); §1.12.4
    public boolean isModal(); §1.12.5
    public boolean isResizable(); §1.12.6
    protected String  paramString(); §1.12.7
    public void setResizable(boolean resizable); §1.12.8
    public void setTitle(String title); §1.12.9
}
```

This class represents a dialog window, a window that takes input from the user.

Dialogs are intended to be temporary windows. They present specific timely information to the user, or they allow the user to specify options for the current operation.

The AWT sends the dialog window all mouse, keyboard, and focus events that occur over it.

By default, the dialog window is invisible. The application must use the show method (II-§1.10.70) to cause the dialog window to appear.

The default layout for a dialog is BorderLayout.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dialog.Dialog

public Dialog(Frame parent, boolean modal)

Creates a dialog window that is initially invisible. If the modal flag is true, the dialog window grabs all input from the user.

The parent argument is the main application window.

Parameters:

parent- the owner of the dialog

modal- if true, the dialog box blocks input to other windows while it is visible

public Dialog(Frame parent, String title, boolean modal)

Creates a titled dialog window that is initially invisible. If the modal flag is true, then the dialog box grabs all input from the user.

The parent argument is the main application window.

Parameters:

parent- the owner of the dialog

title- the title of the dialog

modal- if true, dialog blocks input to other windows when shown

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.addNotify

public void addNotify()

This method calls the createDialog method ([II-§1.41.8](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a DialogPeer ([II-§3.8](#)) for this dialog window. This peer allows the application to change the look of a dialog window without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Window ([II-§1.42.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.getTitle

public String getTitle()

Returns:

the title of this dialog window.

See Also:

setTitle [\(II-§1.12.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.isModal

public boolean isModal()

Returns:

true if this dialog window is modal; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.isResizable

public boolean isResizable()

Indicates whether this dialog window is resizable. By default, a dialog window is resizable.

Returns:

true if the user can resize this dialog window; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.paramString

protected String paramString()

Returns the parameter string representing the state of this dialog window. This string is useful for debugging.

Returns:

the parameter string of this dialog window.

Overrides:

paramString in class Container [\(II-§1.11.16\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dialog.setResizable

public void setResizable(boolean resizable)

Sets the resizable flag for this dialog window.

Parameters:

resizable- true if this dialog window is resizable; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dialog.setTitle

public void setTitle(String title)

Sets the title of this dialog window.

Parameters:

title- the new title

See Also:

getTitle [\(II-§1.12.4\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.13 Class Dimension

```
public class java.awt.Dimension
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public int height; §1.13.1
    public int width; §1.13.2

    // Constructors
    public Dimension(); §1.13.3
    public Dimension(Dimension d); §1.13.4
    public Dimension(int width, int height); §1.13.5

    // Methods
    public String toString(); §1.13.6
}
```

A class that encapsulates the width and height of a component in a single object.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Dimension.height

public int height

The height of the component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dimension.width

public int width

The width of the component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dimension.Dimension

public Dimension()

Creates a Dimension with a width of zero and a height of zero.

public Dimension(Dimension d)

Creates a Dimension whose width and height are the same as the argument.

Parameters:

d- the specified dimension for the width and height values

public Dimension(int width, int height)

Creates a Dimension with the specified width and height.

Parameters:

width- the width

height- the height

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Dimension.toString

public String toString()

Returns:

a string representation of this dimension.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.14 Class Event

```
public class java.awt.Event
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public Object arg; §1.14.1
    public int clickCount; §1.14.2
    public Event evt; §1.14.3
    public int id; §1.14.4
    public int key; §1.14.5
    public int modifiers; §1.14.6
    public Object target; §1.14.7
    public long when; §1.14.8
    public int x; §1.14.9
    public int y; §1.14.10

    // possible values for the id field
    public final static int ACTION_EVENT; §1.14.11
    public final static int GOT_FOCUS; §1.14.12
    public final static int KEY_ACTION; §1.14.13
    public final static int KEY_ACTION_RELEASE; §1.14.14
    public final static int KEY_PRESS; §1.14.15
    public final static int KEY_RELEASE; §1.14.16
    public final static int LIST_DESELECT; §1.14.17
    public final static int LIST_SELECT; §1.14.18
    public final static int LOAD_FILE; §1.14.19
    public final static int LOST_FOCUS; §1.14.20
    public final static int MOUSE_DOWN; §1.14.21
    public final static int MOUSE_DRAG; §1.14.22
    public final static int MOUSE_ENTER; §1.14.23
    public final static int MOUSE_EXIT; §1.14.24
    public final static int MOUSE_MOVE; §1.14.25
    public final static int MOUSE_UP; §1.14.26
    public final static int SAVE_FILE; §1.14.27
    public final static int SCROLL_ABSOLUTE; §1.14.28
    public final static int SCROLL_LINE_DOWN; §1.14.29
    public final static int SCROLL_LINE_UP; §1.14.30
    public final static int SCROLL_PAGE_DOWN; §1.14.31
    public final static int SCROLL_PAGE_UP; §1.14.32
    public final static int WINDOW_DEICONIFY; §1.14.33
    public final static int WINDOW_DESTROY; §1.14.34
    public final static int WINDOW_EXPOSE; §1.14.35
    public final static int WINDOW_ICONIFY; §1.14.36
    public final static int WINDOW_MOVED; §1.14.37
```



```

    // possible values for the key field when the
    // action is KEY_ACTION or KEY_ACTION_RELEASE
    public final static int DOWN;    §1.14.38
    public final static int END;    §1.14.39
    public final static int F1;    §1.14.40
    public final static int F2;    §1.14.41
    public final static int F3;    §1.14.42
    public final static int F4;    §1.14.43
    public final static int F5;    §1.14.44
    public final static int F6;    §1.14.45
    public final static int F7;    §1.14.46
    public final static int F8;    §1.14.47
    public final static int F9;    §1.14.48
    public final static int F10;    §1.14.49
    public final static int F11;    §1.14.50
    public final static int F12;    §1.14.51
    public final static int HOME;    §1.14.52
    public final static int LEFT;    §1.14.53
    public final static int PGDN;    §1.14.54
    public final static int PGUP;    §1.14.55
    public final static int RIGHT;    §1.14.56
    public final static int UP;    §1.14.57

    // possible masks for the modifiers field
    public final static int ALT_MASK    §1.14.58
    public final static int CTRL_MASK;    §1.14.59
    public final static int META_MASK;    §1.14.61
    public final static int SHIFT_MASK;    §1.14.60

    // Constructors
    public Event(Object target, int id, Object arg);    §1.14.62
    public Event(Object target, long when, int id, §1.14.63
        int x, int y, int key, int modifiers);
    public Event(Object target, long when, int id, §1.14.64
        int x, int y, int key,
        int modifiers, Object arg);

    // Methods
    public boolean controlDown();    §1.14.65
    public boolean metaDown();    §1.14.66
    protected String paramString();    §1.14.67
    public boolean shiftDown();    §1.14.68
    public String toString();    §1.14.69
    public void translate(int dx, int dy);    §1.14.70
}

```


Event is a platform-independent class that encapsulates user events from the local Graphical User Interface (GUI) platform.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.arg

public object arg

An arbitrary argument of the event. The value of this field depends on the type of event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.clickCount

public int clickCount

For MOUSE_DOWN events, this field indicates the number of consecutive clicks. For other events, it is zero.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.evt

public **Event** evt

The next event. This field is set when putting events into a linked list.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.id

public int id

The type of the event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.key

public int key

The key that was pressed in a keyboard event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.modifiers

public int modifiers

The state of the modifier keys.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.target

public object target

The target component. This indicates the component over which the event occurred or with which the event is associated.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.when

public long when

The time stamp of the event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.x

public int x

The x coordinate of the event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.y

public int y

The y coordinate of the event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.ACTION_EVENT

```
public final static int ACTION_EVENT = 1001
```

This event indicates that the user wants some action to occur.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.GOT_FOCUS

```
public final static int GOT_FOCUS = 1004
```

A component gained the focus.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.KEY_ACTION

```
public final static int KEY_ACTION = 403
```

The user has pressed an "action" key. The key field contains one of the special values indicated in §1.14.38 to §1.14.57.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.KEY_ACTION_RELEASE

```
public final static int KEY_ACTION_RELEASE = 404
```

The user has released an "action" key. The key field contains one of the special values indicated in §1.14.38 to §1.14.57.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.KEY_PRESS

```
public final static int KEY_PRESS = 401
```

The user has pressed a normal key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.KEY_RELEASE

```
public final static int KEY_RELEASE = 402
```

The user has released a normal key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.LIST_DESELECT

public final static int LIST_DESELECT = 702

An item in a list has been deselected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.LIST_SELECT

```
public final static int LIST_SELECT = 701
```

An item in a list has been selected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.LOAD_FILE

```
public final static int LOAD_FILE = 1002
```

A file loading event.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.LOST_FOCUS

```
public final static int LOST_FOCUS = 1005
```

A component lost the focus.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.MOUSE_DOWN

```
public final static int MOUSE_DOWN = 501
```

The user has pressed the mouse button. The ALT_MASK (II-§1.14.58) flag indicates that the middle button has been pushed. The META_MASK (II-§1.14.61) flag indicates that the right button has been pushed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.MOUSE_DRAG

```
public final static int MOUSE_DRAG = 506
```

The user has moved the mouse with a button pushed. The ALT_MASK ([II-§1.14.58](#)) flag indicates that the middle button has been pushed. The META_MASK ([II-§1.14.61](#)) flag indicates that the right button has been pushed pushed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.MOUSE_ENTER

```
public final static int MOUSE_ENTER = 504
```

The mouse has entered a component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.MOUSE_EXIT

```
public final static int MOUSE_EXIT = 505
```

The mouse has exited a component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.MOUSE_MOVE

public final static int MOUSE_MOVE = 503

The mouse has moved with no button pressed.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.MOUSE_UP

```
public final static int MOUSE_UP = 502
```

The user has released the mouse button. The ALT_MASK ([II-§1.14.58](#)) flag indicates that the middle button has been pushed. The META_MASK ([II-§1.14.61](#)) flag indicates that the right button has been pushed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SAVE_FILE

```
public final static int SAVE_FILE = 1003
```

A file saving event.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SCROLL_ABSOLUTE

```
public final static int SCROLL_ABSOLUTE = 605
```

The user has moved the bubble in a scroll bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SCROLL_LINE_DOWN

public final static int SCROLL_LINE_DOWN = 602

The user has pushed the "line down" area of a scroll bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.SCROLL_LINE_UP

```
public final static int SCROLL_LINE_UP = 601
```

The user has pushed the "line up" area of a scroll bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SCROLL_PAGE_DOWN

```
public final static int SCROLL_PAGE_DOWN = 604
```

The user has pushed the "page down" area of a scroll bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SCROLL_PAGE_UP

public final static int SCROLL_PAGE_UP = 603

The user has pushed the "page up" area of a scroll bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.WINDOW_DEICONIFY

```
public final static int WINDOW_DEICONIFY = 204
```

The user has asked the window manager to deiconify the window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.WINDOW_DESTROY

```
public final static int WINDOW_DESTROY = 201
```

The user has asked the window manager to kill the window.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.WINDOW_EXPOSE

```
public final static int WINDOW_EXPOSE = 202
```

A window has become exposed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.WINDOW_ICONIFY

public final static int WINDOW_ICONIFY = 203

The user has asked the window manager to iconify the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.WINDOW_MOVED

```
public final static int WINDOW_MOVED = 205
```

The window has moved.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.DOWN

```
public final static int DOWN = 1005
```

The down key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.END

```
public final static int END = 1001
```

The end key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F1

```
public final static int F1 = 1008
```

The F1 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F2

```
public final static int F2 = 1009
```

The F2 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F3

```
public final static int F3 = 1010
```

The F3 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F4

```
public final static int F4 = 1011
```

The F4 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F5

```
public final static int F5 = 1012
```

The F5 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F6

```
public final static int F6 = 1013
```

The F6 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F7

```
public final static int F7 = 1014
```

The F7 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F8

```
public final static int F8 = 1015
```

The F8 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F9

```
public final static int F9 = 1016
```

The F9 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F10

```
public final static int F10 = 1017
```

The F10 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F11

```
public final static int F11 = 1018
```

The F11 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.F12

```
public final static int F12 = 1019
```

The F12 function key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.HOME

```
public final static int HOME = 1008
```

The home key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.LEFT

```
public final static int LEFT = 1006
```

The left arrow key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.PGDN

```
public final static int PGDN = 1003
```

The page down key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.PGUP

```
public final static int PGUP = 1002
```

The page up key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.RIGHT

```
public final static int RIGHT = 1007
```

The right arrow key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.UP

```
public final static int UP = 1004
```

The up arrow key.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.ALT_MASK

```
public final static int ALT_MASK = 8
```

This flag indicates that the "alt" key was down when the event occurred. For mouse events, this flag indicates that the middle button was pressed or released.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.CTRL_MASK

```
public final static int CTRL_MASK = 2
```

This flag indicates that the control key was down when the event occurred.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.SHIFT_MASK

```
public final static int SHIFT_MASK = 0
```

This flag indicates that the shift key was down when the event occurred.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.META_MASK

```
public final static int META_MASK = 4
```

This flag indicates that the meta key was down when the event occurred. For mouse events, this flag indicates that the right button was pressed or released.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Event.Event

```
public Event(Object target, int id, Object arg)
```

Creates an Event with the specified target component, event type, and argument.

Parameters:

target- the target component

id- the event type

arg- the specified argument

```
public Event(Object target, long when, int id, int x, int y, int key,  
int modifiers)
```

Creates an Event with the specified target component, time stamp, event type, x and y coordinates, keyboard key, state of the modifier keys, and an argument set to null.

Parameters:

target- the target component

when- the time stamp

id- the event type

x- the x coordinate

y- the y coordinate

key- the key pressed in a keyboard event

modifiers- the state of the modifier keys

```
public Event(Object target, long when, int id, int x, int y, int key,  
int modifiers, Object arg)
```

Creates an Event with the specified target component, time stamp, event type, x and y coordinates, keyboard key, state of the modifier keys, and argument.

Parameters:

target- the target component

when- the time stamp

id- the event type

x- the x coordinate

y- the y coordinate

key- the key pressed in a keyboard event

modifiers- the state of the modifier keys

arg- the specified argument

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.controlDown

public boolean controlDown()

Returns:

true if this event indicates that the control key was down; false otherwise.

See Also:

modifiers [\(II-§1.14.6\)](#)

shiftDown [\(II-§1.14.68\)](#)

metaDown [\(II-§1.14.66\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.metaDown

public boolean metaDown()

Returns:

true if this event indicates that the meta key was down; false otherwise.

See Also:

modifiers [\(II-§1.14.6\)](#)

shiftDown [\(II-§1.14.68\)](#)

controlDown [\(II-§1.14.65\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event paramString

protected String paramString()

Returns the parameter string representing this event. This string is useful for debugging.

Returns:

the parameter string of this event.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.shiftDown

public boolean shiftDown()

Returns:

true if this event indicates that the shift key was down; false otherwise.

See Also:

modifiers [\(II-§1.14.6\)](#)

controlDown [\(II-§1.14.65\)](#)

metaDown [\(II-§1.14.66\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.toString

public String toString()

Returns:

a string representation of this event.

Overrides:

toString in class Object (I-§1.12.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Event.translate

public void translate(int dx, int dy)

Translates this event so that its x and y position are increased by dx and dy respectively.

Parameters:

dx- the amount to translate the x coordinate

dy- the amount to translate the y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.15 Class FileDialog

```
public class java.awt.FileDialog
    extends java.awt.Dialog (II-§1.12)
{
    // Fields
    public final static int LOAD; §1.15.1
    public final static int SAVE; §1.15.2

    // Constructors
    public FileDialog(Frame parent, String title); §1.15.3
    public FileDialog(Frame parent, String title, int mode); §1.15.4

    // Methods
    public void addNotify(); §1.15.5
    public String getDirectory(); §1.15.6
    public String getFile(); §1.15.7
    public FilenameFilter getFilenameFilter(); §1.15.8
    public int getMode(); §1.15.9
    protected String  paramString(); §1.15.10
    public void setDirectory(String dir); §1.15.11
    public void setFile(String file); §1.15.12
    public void setFilenameFilter(FilenameFilter filter); §1.15.13
}
```

The class FileDialog displays a dialog window from which the user can select a file.

Since it is a modal dialog, when its show method (II-§1.42.7) is called, it blocks the rest of the application until the user chooses a file.

The AWT sends the file dialog window all mouse, keyboard, and focus events that occur over it.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDialog.LOAD

```
public final static int LOAD = 0
```

This constant value indicates that the file dialog window is intended to determine a file from which to read.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDialog.SAVE

```
public final static int SAVE = 1
```

This constant value indicates that the file dialog window is intended to determine a file to which to write.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDialog.FileDialog

public FileDialog(Frame parent, String title)

Creates a file dialog window with the specified title for loading a file. The files shown are those in the current directory.

Parameters:

parent- the owner of the dialog

title- the title of the dialog

public FileDialog(Frame parent, String title, int mode)

Creates a file dialog window with the specified title for loading or saving a file.

The mode argument must have the value LOAD (II-§1.15.1) or to SAVE (II-§1.15.2). The value LOAD indicates that the file dialog is finding a file to read. The value SAVE indicates that the file dialog is finding a place to write a file.

Parameters:

parent- the owner of the dialog

title- the title of the dialog

mode- the mode of the dialog

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.addNotify

public void addNotify()

This method calls the createFileDialog method ([II-§1.41.9](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a FileDialogPeer ([II-§3.9](#)) for this file dialog window. The peer allows the application to change the look of a file dialog window without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Dialog ([II-§1.12.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.getDirectory

public String getDirectory()

Returns:

the directory of this dialog.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.getFile

public String getFile()

Returns:

the currently selected file of this file dialog window, or null if none is selected.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.GetFileNameFilter

public FilenameFilter GetFilenameFilter()

Determines this file dialog's filename filter. A filename filter ([I-§2.26](#)) allows the user to specify which files appear in the file dialog window.

Returns:

this file dialog's filename filter.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDialog.getMode

public int getMode()

Indicates whether this file dialog box is for loading from a file or for saving to a file.

Returns:

the mode of this file dialog window; the value is either LOAD (II-§1.15.1) or SAVE (II-§1.15.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.paramString

protected String paramString()

Returns the parameter string representing the state of this file dialog window. This string is useful for debugging.

Returns:

the parameter string of this file dialog window.

Overrides:

paramString in class Dialog ([II-§1.12.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.setDirectory

public void setDirectory(String dir)

Sets the directory of this file dialog window to be the specified directory.

Parameters:

dir- the specific directory

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.setFile

public void setFile(String file)

Sets the selected file for this file dialog window to be the specified file. This file becomes the default file if it is set before the file dialog window is first shown.

Parameters:

file- the file being set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialog.setFilenameFilter

public void setFilenameFilter(FilenameFilter filter)

Sets the filename filter ([I-§2.26](#)) for this file dialog window to the specified filter.

Parameters:

`filter`- the specified filter

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.16 Class FlowLayout

```
public class java.awt.FlowLayout
    extends java.lang.Object (I-§1.12)
    implements java.awt.LayoutManager (II-§1.43)
{
    // Fields
    public final static int CENTER; §1.16.1
    public final static int LEFT; §1.16.2
    public final static int RIGHT; §1.16.3

    // Constructors
    public FlowLayout(); §1.16.4
    public FlowLayout(int align); §1.16.5
    public FlowLayout(int align, int hgap, int vgap); §1.16.6

    // Methods
    public void addLayoutComponent(String name, §1.16.7
                                   Component comp);
    public void layoutContainer(Container target); §1.16.8
    public Dimension minimumLayoutSize(Container target); §1.16.9
    public Dimension preferredLayoutSize(Container target); §1.16.10
    public void removeLayoutComponent(Component comp); §1.16.11
    public String toString(); §1.16.12
}
```

A Flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel.

For example, the following picture shows an Applet using the flow layout manager (its default layout manager) to position three buttons{ewc msdncl, EWGraphic, AWT0em 0 /a "sunref.BMP"}:

Here is the applet code:

```
import java.awt.*;
import java.applet.Applet;

public class myButtons extends Applet {
    Button button1, button2, button3;
    public void init() {
        button1 = new Button("Ok");
        button2 = new Button("Open");
        button3 = new Button("Close");
        add(button1);
        add(button2);
        add(button3);
    }
}
```


A flow layout lets each component take its natural (preferred) size.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FlowLayout.CENTER

```
public final static int CENTER = 1
```

This value indicates that each row of components should be centered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FlowLayout.LEFT

```
public final static int LEFT = 0
```

This value indicates that each row of components should be left justified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FlowLayout.RIGHT

```
public final static int RIGHT = 2
```

This value indicates that each row of components should be right justified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FlowLayout.FlowLayout

public FlowLayout()

Creates a new flow layout manager with a centered alignment and a default 5-pixel horizontal and vertical gap.

public FlowLayout(int align)

Creates a new flow layout manager with the indicated alignment and a default 5-pixel horizontal and vertical gap.

The alignment argument must be one of FlowLayout.LEFT, FlowLayout.RIGHT, or FlowLayout.CENTER.

Parameters:

align- the alignment value

public FlowLayout(int align, int hgap, int vgap)

Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

The alignment argument must be one of FlowLayout.LEFT, FlowLayout.RIGHT, or FlowLayout.CENTER.

Parameters:

align- the alignment value

hgap- the horizontal gap between components

vgap- the vertical gap between components

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.addLayoutComponent

public void addLayoutComponent(String name, Component comp)

This method is not used by the flow layout manager.

Parameters:

name- a tag

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.layoutContainer

public void layoutContainer(Container target)

Lays out the container argument using this layout.

This method lets each component take its preferred size.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target– the container in which to do the layout

See Also:

Container (II-§1.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.minimumLayoutSize

public Dimension minimumLayoutSize(Container target)

Determines the minimum size of the target container using this flow layout.

The minimum width needed to lay out the container's components is the total minimum width of each of the components, plus (ncomponents + 1) times the horizontal gap, plus the left and right inset, where ncomponents is the number of components in the container.

The minimum height needed to lay out the container's components is the greatest minimum height of the components, plus twice the vertical gap, plus the top and bottom insets.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target– the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

preferredLayoutSize (II-§1.16.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.preferredLayoutSize

public Dimension preferredLayoutSize(Container target)

Determines the preferred size of the target container using this flow layout.

The preferred width to lay out the container's components is the total preferred width of each of the components, plus (ncomponents + 1) times the horizontal gap, plus the left and right inset, where ncomponents is the number of components in the container.

The preferred height to lay out the container's components is the greatest preferred height of the components, plus twice the vertical gap, plus the top and bottom insets.

Most applications do not call this method directly. This method is called when a container calls its preferredSize method (II-§1.11.17).

Parameters:

parent - the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

Container (II-§1.11)

minimumLayoutSize (II-§1.16.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.removeLayoutComponent

public void removeLayoutComponent(Component comp)

Removes the specified component from this layout.

Most applications do not call this method directly. This method is called when a container calls its remove (II-§1.11.19) or removeAll (II-§1.11.20) methods.

Parameters:

comp- the component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FlowLayout.toString

public String toString()

Returns:

a string representation of this layout.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.17 Class Font

```
public class java.awt.Font
    extends java.lang.Object (l-§1.12)
{
    // Fields
    protected String name; §1.17.1
    protected int size; §1.17.2
    protected int style; §1.17.3

    // style has the following bit masks
    public final static int BOLD; §1.17.4
    public final static int ITALIC; §1.17.5
    public final static int PLAIN §1.17.6

    // Constructors
    public Font(String name, int style, int size); §1.17.7

    // Methods
    public boolean equals(Object obj); §1.17.8
    public String getFamily(); §1.17.9
    public static Font getFont(String nm); §1.17.10
    public static Font getFont(String nm, Font font); §1.17.11
    public String getName(); §1.17.12
    public int getSize(); §1.17.13
    public int getStyle(); §1.17.14
    public int hashCode(); §1.17.15
    public boolean isBold(); §1.17.16
    public boolean isItalic(); §1.17.17
    public boolean isPlain(); §1.17.18
    public String toString(); §1.17.19
}
```

This class represents a font.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.name

protected String name

The logical name of this font.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.size

protected int size

The point size of this font.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.style

protected int style

The style of the font. This is the sum of the constants PLAIN, BOLD, or ITALIC.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.BOLD

```
public final static int BOLD = 1
```

The bold style constant. This can be combined with the other style constants for mixed styles.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Font.ITALIC

```
public final static int ITALIC = 2
```

The italicized style constant. This can be combined with the other style constants for mixed styles.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Font.PLAIN

```
public final static int PLAIN = 0
```

The plain style constant. This can be combined with the other style constants for mixed styles.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Font.Font

public Font(String name, int style, int size)

Creates a new font with the specified name, style, and point size.

Parameters:

name- the font name

style- the constant style used

size- the point size of the font

See Also:

getFontList in class Toolkit ([II-§1.41.24](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.equals

public boolean equals (Object obj)

The result is true if the argument is not null and is a Font object with the same name, same style, and same point size as this font.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object (I-§1.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.getFamily

public String getFamily()

Returns:

the platform-specific family name of this font.

See Also:

getName [\(II-§1.17.12\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.getFont

`public static Font getFont(String nm)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (I-§1.18.10). The string value of this property is then interpreted as a font as described in the `getFont` method of two argument (II-§1.17.11).

If the specified property is not found, null is returned.

Parameters:

`nm`– the property name

Returns:

the font value of the property.

`public static Font getFont(String nm, Font font)`

The first argument is treated as the name of a system property to be obtained as if by the method `System.getProperty` (I-§1.18.10). The string value of this property is then interpreted as a font.

The property value should be one of the following forms:

fontname-style-pointsize
fontname-pointsize
fontname-style
fontname

where *style* is one of the three strings "bold", "bolditalic" or "italic" and *pointsize* is a decimal representation of the *pointsize*.

The default style is PLAIN. The default *pointsize* is 12.

If the specified property is not found, the font argument is returned instead.

Parameters:

`nm`– the property name

`font`– a default font to return if the property isn't defined

Returns:

the font value of the property.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Font.getName

public String getName()

Returns:

the logical name of this font.

See Also:

getFamily [\(II-§1.17.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.GetSize

```
public int GetSize()
```

Returns:

the point size of this font.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Font.getStyle

public int `getStyle()`

Returns:

the style of this font.

See Also:

isPlain [\(II-§1.17.18\)](#)

isBold [\(II-§1.17.16\)](#)

isItalic [\(II-§1.17.17\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.hashCode

public int hashCode()

Returns:

a hash code value for this font.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.isBold

public boolean isBold()

Returns:

true if this font is bold; false otherwise.

See Also:

getStyle [\(II-§1.17.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.IsItalic

public boolean IsItalic()

Returns:

true if this font is italic; false otherwise.

See Also:

getStyle [\(II-§1.17.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.isPlain

public boolean isPlain()

Returns:

true if this font is neither bold nor italic; false otherwise.

See Also:

getStyle [\(II-§1.17.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Font.toString

public String toString()

Returns:

a string representation of this font.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.18 Class FontMetrics

```
public abstract class java.awt.FontMetrics
    extends java.lang.Object (I-§1.12)
{
    // Fields
    protected Font font; §1.18.1

    // Constructors
    protected FontMetrics(Font font); §1.18.2

    // Methods
    public int bytesWidth(byte data[], int off, int len); §1.18.3
    public int charsWidth(char data[], int off, int len); §1.18.4
    public int charWidth(char ch); §1.18.5
    public int charWidth(int ch); §1.18.6
    public int getAscent(); §1.18.7
    public int getDescent(); §1.18.8
    public Font getFont(); §1.18.9
    public int getHeight(); §1.18.10
    public int getLeading(); §1.18.11
    public int getMaxAdvance(); §1.18.12
    public int getMaxAscent(); §1.18.13
    public int getMaxDescent(); §1.18.14
    public int[] getWidths(); §1.18.15
    public int stringWidth(String str); §1.18.16
    public String toString(); §1.18.17
}
```

This class represents a font metrics object, which gives information about the rendering of a particular font on a particular screen.

When {ewc msdn cd, EWGraphic, AWT0 eo 0 /a "sunref.BMP"} an application asks the AWT to place a character at the position (x, y) , the character is placed so that its reference point (shown as the dot in the picture) is put at that position. The reference point specifies a horizontal line called the *baseline* of the character. In normal printing, the baselines of the characters should align. In addition, every character in a font has an *ascent*, a *descent*, and an *advance width*. The ascent is the amount by which the character ascends above the baseline. The descent is the amount by which the character descends below the baseline.

The advance width indicates the position at which AWT should place the next character. If the current character is placed with its reference point at the position (x, y) , and the character's advance width is w , then the following character is placed with its reference point at the position $(x + w, y)$. The advance width is often the same as the width of character's bounding box, but need not be so. In particular, slanted and italic fonts often have characters whose top-right corner extends slightly beyond the advance width.

An array of characters or a string can also have an ascent, a descent, and an advance width. The ascent of the array is the maximum ascent of any character in the array. The descent is the

maximum descent of any character in the array. The advance width is the sum of the advance widths of each of the characters in the array.

The default implementations of these methods are inefficient; they are usually overridden with more efficient toolkit-specific implementations.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FontMetrics.font

protected Font font

The actual font.

See Also:

getFont [\(II-§1.18.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.FontMetrics

protected **FontMetrics**(**Font** font)

Creates a new FontMetrics object for finding out height and width information about the specified font and specific character glyphs in that font.

Parameters:

font- the font

See Also:

Font (II-§1.17).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FontMetrics.bytesWidth

public int bytesWidth(byte data[], int off, int len)

Parameters:

- data- an array of characters
- off- the start offset of the data
- len- the number of bytes to be measured

Returns:

the advance width of the subarray of the specified byte array in the font described by this font metric.

See Also:

- stringWidth [\(II-§1.18.16\)](#)
- charsWidth [\(II-§1.18.4\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.charsWidth

public int charsWidth(char data[], int off, int len)

Parameters:

`data`- an array of characters
`off`- the start offset of the data
`len`- the number of bytes measured

Returns:

the advance width of the subarray of the specified char array in the font described by this font metric.

See Also:

`stringWidth` [\(II-§1.18.16\)](#)
`bytesWidth` [\(II-§1.18.3\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.charWidth

public int charWidth(char ch)

Parameters:

ch- a char

Returns:

the advance width of the specified char in the font described by this font metric.

See Also:

stringWidth [\(II-§1.18.16\)](#).

public int charWidth(int ch)

Parameters:

ch- a char

Returns:

the advance width of the specified character in the font described by this font metric.

See Also:

stringWidth [\(II-§1.18.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getAscent

public int `getAscent()`

Determines the *font ascent* of the font described by this font metric. The font ascent is the distance from the base line to the top of most alphanumeric characters. Some characters in the font may extend above this distance.

Returns:

the font ascent of the font.

See Also:

`getMaxAscent` [\(II-§1.18.13\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getDescent

public int getDescent()

Determines the *font descent* of the font described this font metric. The font descent is the distance from the base line to the bottom of most alphanumeric characters. Some characters in the font may extend below this distance.

Returns:

the font descent of the font.

See Also:

getMaxDescent [\(II-§1.18.14\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getFont

public **Font** **getFont()**

Returns:

the font described by this font metric.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FontMetrics.getHeight

public int getHeight()

Determines the *standard height* of a line of text in the font described by this font metric. This standard height is the distance between the baseline of adjacent lines of text. It is computed as the sum:

`getLeading() + getAscent() + getDescent()`

There is no guarantee that lines of text spaced at this distance must be disjoint; such lines may overlap if some characters overshoot either the standard ascent or the standard descent.

Returns:

the standard height of the font.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getLeading

public int getLeading()

Determines the *standard leading* of the font described by this font metric. The standard leading (inter-line spacing) is the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line. The height metric is calculated to include this extra space.

Returns:

the standard leading of the font.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getMaxAdvance

public int ~~getMaxAdvance~~()

Returns:

the maximum advance width of any character in the font, or -1 if the maximum advance is not known.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getMaxAscent

public int ~~getMax~~Ascent()

Determines the maximum ascent of the font described by this font metric. No character extends further above the baseline than this height.

Returns:

the maximum ascent of any character in the font.

See Also:

getAscent [\(II-§1.18.7\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getMaxDescent

public int getMaxDescent()

Determines the maximum descent of the font described by this font metric. No character extends further below the baseline than this height.

Returns:

the maximum descent of any character in the font.

See Also:

getDescent (II-§1.18.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.getWidths

public int[] getWidths()

Returns:

any array giving the advance widths of the first 256 characters in the font described by this font metric.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.stringWidth

public int stringWidth(String str)

Parameters:

str- a string

Returns:

the advance width of the specified string in the font described by this font metric.

See Also:

charsWidth [\(II-§1.18.4\)](#)

bytesWidth [\(II-§1.18.3\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FontMetrics.toString

public String toString()

Returns:

a string representation of this font metric.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.19 Class Frame

```
public class java.awt.Frame
    extends java.awt.Window (II-§1.42)
    implements java.awt.MenuContainer (II-§1.44)
{
    // possible cursor types for the setCursor method
    public final static int CROSSHAIR_CURSOR; §1.19.1
    public final static int DEFAULT_CURSOR; §1.19.2
    public final static int E_RESIZE_CURSOR; §1.19.3
    public final static int HAND_CURSOR; §1.19.4
    public final static int MOVE_CURSOR; §1.19.5
    public final static int N_RESIZE_CURSOR; §1.19.6
    public final static int NE_RESIZE_CURSOR; §1.19.7
    public final static int NW_RESIZE_CURSOR; §1.19.8
    public final static int S_RESIZE_CURSOR; §1.19.9
    public final static int SE_RESIZE_CURSOR; §1.19.10
    public final static int SW_RESIZE_CURSOR; §1.19.11
    public final static int TEXT_CURSOR; §1.19.12
    public final static int W_RESIZE_CURSOR; §1.19.13
    public final static int WAIT_CURSOR; §1.19.14

    // Constructors
    public Frame(); §1.19.15
    public Frame(String title); §1.19.16

    // Methods
    public void addNotify(); §1.19.17
    public void dispose(); §1.19.18
    public int getCursorType(); §1.19.19
    public Image getIconImage(); §1.19.20
    public MenuBar getMenuBar(); §1.19.21
    public String getTitle(); §1.19.22
    public boolean isResizable(); §1.19.23
    protected String  paramString(); §1.19.24
    public void remove(MenuComponent m); §1.19.25
    public void setCursor(int cursorType); §1.19.26
    public void setIconImage(Image image); §1.19.27
    public void setMenuBar(MenuBar mb); §1.19.28
    public void setResizable(boolean resizable); §1.19.29
    public void setTitle(String title); §1.19.30
}
```

A frame is a top-level window with a title and a border. A frame can also have a menu bar. The AWT sends the frame all mouse, keyboard, and focus events that occur over it.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.CROSSHAIR_CURSOR

```
public final static int CROSSHAIR_CURSOR = 1
```

A cross-haired shaped cursor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.DEFAULT_CURSOR

```
public final static int DEFAULT_CURSOR = 0
```

The default cursor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.E_RESIZE_CURSOR

```
public final static int E_RESIZE_CURSOR = 11
```

A cursor indicating that the right-hand border is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.HAND_CURSOR

```
public final static int HAND_CURSOR = 12
```

A hand-shaped cursor.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.MOVE_CURSOR

```
public final static int MOVE_CURSOR = 13
```

A cursor indicating that an object is being moved.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.N_RESIZE_CURSOR

```
public final static int N_RESIZE_CURSOR = 8
```

A cursor indicating that the top border is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.NE_RESIZE_CURSOR

```
public final static int NE_RESIZE_CURSOR = 7
```

A cursor indicating that the top-right corner is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.NW_RESIZE_CURSOR

```
public final static int NW_RESIZE_CURSOR = 6
```

A cursor indicating the the top-left corner is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.S_RESIZE_CURSOR

```
public final static int S_RESIZE_CURSOR = 9
```

A cursor indicating the the bottom edge is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.SE_RESIZE_CURSOR

```
public final static int SE_RESIZE_CURSOR = 5
```

A cursor indicating that the bottom-right edge is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.SW_RESIZE_CURSOR

```
public final static int SW_RESIZE_CURSOR = 4
```

A cursor indicating that the bottom-left edge is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.TEXT_CURSOR

```
public final static int TEXT_CURSOR = 2
```

A cursor for text insertion.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.W_RESIZE_CURSOR

```
public final static int W_RESIZE_CURSOR = 10
```

A cursor indicating the the left edge is being resized.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.WAIT_CURSOR

```
public final static int WAIT_CURSOR = 3
```

A cursor indicating that a long-running operation is taking place.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.Frame

public **Frame**()

Constructs a new frame; it is initially invisible and has no title.

public **Frame**(String title)

Constructs a new frame; it is initially invisible and has the specified title.

Parameters:

title- the title

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Frame.addNotify

public void addNotify()

This method calls the createFrame method (II-§1.41.10) of this object's toolkit (II-§1.10.20) in order to create a FramePeer (II-§3.10) for this frame. This peer allows the application to change the look of a frame without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Window (II-§1.42.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.dispose

public void dispose()

Disposes of this frame, its menu bar, and any system resources used by this frame.

Overrides:

dispose in class Window (II-§1.42.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.getCursorType

public int getCursorType()

Returns:

the cursor type of this frame.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.getIconImage

public Image getIconImage()

Returns:

the icon image for this frame, or null if this frame doesn't have an icon image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.getMenuBar

public **MenuBar** getMenuBar ()

Returns:

the menu bar for this frame, or null if this frame doesn't have a menu bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.getTitle

public String getTitle()

Returns:

the title of this frame, or null if this frame doesn't have a title.

See Also:

setTitle [\(II-§1.19.30\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.isResizable

public boolean isResizable()

Indicates whether this frame is resizable. By default, all frames are resizable.

Returns:

true if the user can resize this frame; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame paramString

protected String paramString()

Returns the parameter string representing the state of this frame. This string is useful for debugging.

Returns:

the parameter string of this frame.

Overrides:

paramString in class Container (II-§1.11.16).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.remove

public void remove(MenuComponent m)

Removes the specified menu bar from this frame.

Parameters:

m – the menu component to remove

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.setCursor

public void setCursor(int cursorType)

Sets the cursor image for this frame to be one of the predefined cursors.

Parameters:

cursorType- one of the predefined cursor constants ([II-\\$1.19.1-\\$1.19.14](#))

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.setIconImage

public void setIconImage(Image image)

Sets the image to display when this frame is iconized. The default icon is platform specific. Not all platforms support the concept of iconizing a window.

Parameters:

image- the icon image to be displayed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.setMenuBar

public void setMenuBar(MenuBar mb)

Sets the menu bar of this frame to the specified menu bar.

Parameters:

mb – the new menu bar

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.setResizable

public void setResizable(boolean resizable)

Determines whether this frame should be resizable. By default, a frame is resizable.

Parameters:

resizable- true if this frame should be resizable; false otherwise

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Frame.setTitle

public void setTitle(String title)

Sets the title of this frame to the specified title.

Parameters:

title- the new title of this frame, or null to remove the title

See Also:

getTitle [\(II-§1.19.22\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.20 Class Graphics

```
public abstract class java.awt.Graphics
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    protected Graphics() §1.20.1

    // Methods
    public abstract void clearRect(int x, int y, §1.20.2
                                   int width, int height);
    public abstract void clipRect(int x, int y, §1.20.3
                                   int width, int height);

    public abstract void §1.20.4
        copyArea(int x, int y, int width,
                  int height, int dx, int dy);
    public abstract Graphics create() §1.20.5
    public Graphics create(int x, int y, §1.20.6
                          int width, int height);
    public abstract void dispose() §1.20.7
    public void draw3DRect(int x, int y, int width, §1.20.8
                          int height, boolean raised);
    public abstract void drawArc(int x, int y, int width, §1.20.9
                                   int height, int
startAngle,
                                   int arcAngle);
    public void drawBytes(byte data[], int offset, §1.20.10
                          int length, int x, int y);
    public void drawChars(char data[], int offset, §1.20.11
                          int length, int x, int y);
    public abstract boolean §1.20.12
        drawImage(Image img, int x, int y, Color bgcolor,
                   ImageObserver observer);
    public abstract boolean §1.20.13
        drawImage(Image img, int x, int y,
                   ImageObserver observer);
    public abstract boolean §1.20.14
        drawImage(Image img, int x, int y,
                   int width, int height, Color bgcolor,
                   ImageObserver observer);
    public abstract boolean §1.20.15
        drawImage(Image img, int x, int y,
                   int width, int height,
                   ImageObserver observer);
    public abstract void drawLine(int x1, int y1, §1.20.16
                                   int x2, int y2);
    public abstract void drawOval(int x, int y, §1.20.17
                                   int width, int height);
    public abstract void §1.20.18
        drawPolygon(int xPoints[], int yPoints[],
```



```

        int nPoints);
    public void drawPolygon(Polygon p); §1.20.19
    public void drawRect(int x, int y, §1.20.20
        int width, int height);
    public abstract void §1.20.21
        drawRoundRect(int x, int y, int width,
            int height, int arcWidth,
            int arcHeight);
    public abstract void §1.20.22
        drawString(String str, int x, int y);
    public void §1.20.23
        fill3DRect(int x, int y, int width,
            int height, boolean raised);
    public abstract void §1.20.24
        fillArc(int x, int y, int width,
            int height, int startAngle,
            int arcAngle);
    public abstract void §1.20.25
        fillOval(int x, int y, int width, int height);
    public abstract void §1.20.26
        fillPolygon(int xPoints[], int yPoints[], int nPoints);
    public void fillPolygon(Polygon p); §1.20.27
    public abstract void §1.20.28
        fillRect(int x, int y, int width, int height);
    public abstract void §1.20.29
        fillRoundRect(int x, int y, int width, int height,
            int arcWidth, int arcHeight);
    public void finalize(); §1.20.30
    public abstract Rectangle getClipRect(); §1.20.31
    public abstract Color getColor(); §1.20.32
    public abstract Font getFont(); §1.20.33
    public FontMetrics getFontMetrics(); §1.20.34
    public abstract FontMetrics getFontMetrics(Font f); §1.20.35
    public abstract void setColor(Color c); §1.20.36
    public abstract void setFont(Font font); §1.20.37
    public abstract void setPaintMode(); §1.20.38
    public abstract void setXORMode(Color c1); §1.20.39
    public String toString(); §1.20.40
    public abstract void translate(int x, int y); §1.20.41
}

```

The Graphics class is the abstract base class for all graphics contexts which allow an application to draw onto components or onto off-screen images.

Unless otherwise stated, all graphics operations performed with a graphics context object only modify bits within the graphic context's clipping region (II-§1.20.3). All drawing or writing is done in the current color (II-§1.20.36) using the current paint mode (II-§1.20.38, §1.20.39) and in the current font (II-§1.20.37).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Graphics.Graphics

protected Graphics()

This constructor is the default constructor for a graphics context.

Since Graphics is an abstract class, applications cannot call this constructor directly. Graphics contexts are obtained from other graphics contexts [\(II-§1.20.5\)](#) or are created by a component [\(II-§1.10.17\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.clearRect

public abstract void

clearRect(int x, int y, int width, int height)

Clears the specified rectangle by filling it with the background color of the screen¹. This operation does not use the current paint mode (II-§1.20.38, §1.20.39).

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

See Also:

fillRect (II-§1.20.28)

drawRect (II-§1.20.20).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.clipRect

public abstract void

clipRect(int x, int y, int width, int height)

Sets a clipping rectangle for this graphics context. The resulting clipping area is the intersection of the current clipping area and the specified rectangle.

Graphics operations performed with this graphics context have no effect outside the clipping area.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

See Also:

getClipRect ([II-§1.20.31](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.copyArea

public abstract void

copyArea(int x, int y, int width, int height, int dx, int dy)

Copies an area. The source is the rectangle with origin (x, y), and size specified by the width and height arguments. The destination is the rectangle with origin (x + dx, y + dy) and the same width and height.

The clipping rectangle of this graphics context affects only the destination, not the source.

Parameters:

x- the x-coordinate of the source

y- the y-coordinate of the source

width- the width

height- the height

dx- the horizontal distance

dy- the vertical distance

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.create

public abstract Graphics create()

Returns:

a new graphics context that is a copy of this graphics context.

public Graphics

create(int x, int y, int width, int height)

Creates a new graphics context. The new graphics context is identical to this graphics context, except in two respects:

- The new context is translated by (x, y). That is to say, the point (x, y) in the new graphics context is the same as (x, y) in this graphics context.
- The graphics context has an additional clipping rectangle with origin (0, 0), and size specified by the width and height arguments, in addition to whatever (translated) clipping rectangle it inherited from this graphics context.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the clipping rectangle

height- the height of the clipping rectangle

Returns:

a new graphics context.

See Also:

translate ([II-§1.20.41](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.dispose

public **abstract void dispose()**

Disposes of this graphics context.

The graphics context cannot be used after being disposed.

See Also:

finalize [\(II-§1.20.30\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.draw3DRect

```
public void draw3DRect(int x, int y, int width, int height, boolean raised)
```

Draws a highlighted 3-D rectangle.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

raised- if true, the rectangle appears raised; if false, it appears lowered

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Graphics.drawArc

public abstract void

`drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`

Draws a single circular or elliptical arc.

The center of the arc is the center of the rectangle whose origin is (x, y) and whose size is specified by the width and height arguments.

The two axes of the arc are given by the width and height arguments.

The arc is drawn from startAngle to startAngle + arcAngle. The start angle and arc angle are in degrees, not radians.

A start angle of 0 indicates the 3-o'clock position. A positive arc angle indicates a counter-clockwise rotation; a negative arc angle indicates a clockwise rotation.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

startAngle- the beginning angle

arcAngle- the angle of the arc (relative to startAngle)

See Also:

fillArc ([II-§1.20.24](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawBytes

public void drawBytes(byte data[], int offset,int length, int x, int y)

Draws the text given by the specified byte array using this graphics context's current font and color. The baseline of the first character is at position (x, y) in this graphics context's coordinate system.

Parameters:

data- the data to be drawn

offset- the start offset in the data

length- the number of bytes that are drawn

x- the x coordinate

y- the y coordinate

See Also:

drawString ([II-§1.20.22](#))

drawChars ([II-§1.20.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawChars

public void drawChars(char data[], int offset, int length, int x, int y)

Draws the text given by the specified character array using this graphics context's current font and color. The baseline of the first character is at position (x, y) in the graphics context's coordinate system.

Parameters:

data- the array of characters to be drawn
offset- the start offset in the data
length- the number of characters to be drawn
x- the x coordinate
y- the y coordinate

See Also:

drawString [\(II-§1.20.22\)](#)
drawBytes [\(II-§1.20.10\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawImage

public abstract boolean

drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)

Draws the specified image with its top left corner at (x, y) in this graphics context's coordinate space. Transparent pixels in the image are drawn in the specified background color.

If the image has not yet been completely loaded, the image observer's imageUpdate ([II-§2.11.9](#)) method is notified as more of the image becomes available.

Parameters:

img- the specified image to be drawn

x- the x coordinate

y- the y coordinate

bgcolor- the background color

observer- object to notify when the image is loaded.

Returns:

true if all bits of the image are available; false otherwise.

See Also:

Image ([II-§1.24](#))

ImageObserver ([II-§2.11](#)).

public abstract boolean

drawImage(Image img, int x, int y, ImageObserver observer)

Draws the specified image with its top left corner at (x, y) in this graphics context's coordinate space. Transparent pixels in the image do not affect whatever pixels are already there.

If the image has not yet been completely loaded, the image observer's imageUpdate ([II-§2.11.9](#)) method is notified as more of the image becomes available.

Parameters:

`img`- the specified image to be drawn
`x`- the x coordinate
`y`- the y coordinate
`observer`- object to notify when the image is loaded

Returns:

true if all bits of the image are available; false otherwise.

See Also:

Image [\(II-§1.24\)](#)
ImageObserver [\(II-§2.11\)](#).

public abstract boolean

`drawImage(Image img, int x, int y, int width, int height, Color
bgcolor, ImageObserver observer)`

Draws the specified image inside the specified rectangle of this graphics context's coordinate space. The image is scaled if necessary. Transparent pixels in the image are drawn in the specified background color.

If the image has not yet been completely loaded, the image observer's `imageUpdate` [\(II-§2.11.9\) method](#) is notified as more of the image becomes available.

Parameters:

`img`- the specified image to be drawn
`x`- the x coordinate
`y`- the y coordinate
`width`- the width of the rectangle
`height`- the height of the rectangle
`bgcolor`- background color for the image
`observer`- object to notify when the image is loaded

Returns:

true if all bits of the image are available; false otherwise.

See Also:

Image [\(II-§1.24\)](#)
ImageObserver [\(II-§2.11\)](#).

public abstract boolean

drawImage(Image img, int x, int y, int width, int height,
ImageObserver observer)

Draws the specified image inside the specified rectangle of this graphics context's coordinate space. The image is scaled if necessary. Transparent pixels in the image do not affect whatever pixels are already there.

If the image has not yet been completely loaded, the image observer's `imageUpdate` ([II-§2.11.9](#)) method is notified as more of the image becomes available.

Parameters:

`img`- the specified image to be drawn

`x`- the x coordinate

`y`- the y coordinate

`width`- the width of the rectangle

`bgcolor`- background color for the image

`observer`- object to notify when the image is loaded

Returns:

true if all bits of the image are available; false otherwise.

See Also:

Image ([II-§1.24](#))

ImageObserver ([II-§2.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawLine

public abstract void

drawLine(int x1, int y1, int x2, int y2)

Draws a line from (x1, y1) to (x2, y2) in this graphics context's coordinate system.

The line is drawn below and to the right of the logical coordinates.

Parameters:

x1– the first point's x coordinate

y1– the first point's y coordinate

x2– the second point's x coordinate

y2– the second point's y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawOval

public abstract void

drawOval(int x, int y, int width, int height)

Draws a circle or an ellipse such that it fits within the rectangle specified by the x, y, width and height arguments.

The center of the oval is the center of the rectangle whose origin is (x, y) is this graphics context's coordinate system and whose size is specified by the width and height arguments.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

See Also:

fillOval ([II-§1.20.25](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawPolygon

public abstract void

drawPolygon(int xPoints[], int yPoints[], int nPoints)

Draws a closed polygon defined by an array of x points and y points.

This method draws the polygon defined by npoint line segments, where the first npoint-1 line segments are lines segments from (xpoints[i - 1], ypoints[i - 1]) to (xpoints[i], ypoints[i]), for 1 is less than or equal to i is less than npoints. The last line segment starts at the final point and ends at the first point.

Parameters:

xPoints- an array of x points

yPoints- an array of y points

nPoints- the total number of points

See Also:

fillPolygon (II-§1.20.26).

public void drawPolygon(Polygon p)

Draws a polygon defined by the specified polygon argument.

Parameters:

p- a polygon

See Also:

fillPolygon (II-§1.20.26).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawRect

public void drawRect(int x, int y, int width, int height)

Draws the outline of the specified rectangle using the current color. The left and right edges of the rectangle are at `x` and `x + width` respectively. The top and bottom edges of the rectangle are at `y` and `y + height` respectively.

Parameters:

`x`- the x coordinate

`y`- the y coordinate

`width`- the width of the rectangle

`height`- the height of the rectangle

See Also:

`fillRect` ([II-§1.20.28](#))

`clearRect` ([II-§1.20.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawRect

public abstract void

drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

Draws an outlined round-cornered rectangle using this graphics context's current color. The left and right edges of the rectangle are at x and x + width respectively. The top and bottom edges of the rectangle are at y and y + height.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

arcWidth- the horizontal diameter of the arc at the four corners

arcHeight- the vertical diameter of the arc at the four corners

See Also:

fillRoundRect [\(II-§1.20.29\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.drawString

public abstract void drawString(String str, int x, int y)

Draws the string using this graphics context's current font and color. The baseline of the first character is at position (x, y) in the graphics context's coordinate system.

Parameters:

str- the string to be drawn

x- the x coordinate

y- the y coordinate

See Also:

drawChars [\(II-§1.20.11\)](#)

drawBytes [\(II-§1.20.10\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.fill3DRect

```
public void fill3DRect(int x, int y, int width, int height, boolean raised)
```

Draws a highlighted 3-D rectangle that is filled with this graphics context's current color.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

raised- if true, the rectangle is raised; if false, it is lowered

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Graphics.fillArc

public abstract void

fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Draws a single circular or elliptical arc that is filled with this graphics context's current color. The result is a pie shape.

The center of the arc is the center of the rectangle whose origin is (x, y) and whose size is specified by the width and height arguments.

The two axes of the arc are given by the width and height arguments.

The arc is drawn from startAngle to startAngle + arcAngle. The start angle and arc angle are in degrees, not radians.

A start angle of 0 indicates the 3-o'clock position. A positive arc angle indicates a counter-clockwise rotation; a negative arc angle indicates a clockwise rotation.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

startAngle- the beginning angle

arcAngle- the angle of the arc (relative to startAngle)

See Also:

drawArc ([II-§1.20.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.fillOval

public abstract void

fillOval(int x, int y, int width, int height)

Draws a filled circle or a filled ellipse using this graphics context's current color.

The center of the oval is the center of the rectangle whose origin is (x, y) is the graphics context's coordinate system and whose size is specified by the width and height arguments.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

See Also:

drawOval ([II-§1.20.17](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.fillPolygon

public abstract void

fillPolygon(int xPoints[], int yPoints[], int nPoints)

Fills a polygon defined by an array of x points and y points with this graphics context's current color.

This method fills the polygon defined by npoint line segments, where the first npoint-1 line segments are lines segments from (xpoints[i - 1], ypoints[i - 1]) to (xpoints[i], ypoints[i]), for i is less than or equal to npoints. The last line segment starts at the final point and ends at the first point.

The area inside the polygon is defined using an "even-odd" fill rule, also known as the "alternating rule."

Parameters:

xPoints- an array of x points

yPoints- an array of y points

nPoints- the total number of points

See Also:

drawPolygon ([II-§1.20.18](#)).

public void fillPolygon(Polygon p)

Fills a polygon defined by the specified polygon argument with this graphics context's current color.

The area inside the polygon is defined using an "even-odd" fill rule, also known as the "alternating rule."

Parameters:

p- the polygon

See Also:

drawPolygon (II-§1.20.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.fillRect

public **abstract** **void**

fillRect(int x, int y, int width, int height)

Fills the specified rectangle with this graphics context's current color. The left and right edges are at `x` and `x + width - 1` respectively. The top and bottom edges are at `y` and `y + height - 1` respectively.

Parameters:

`x`- the x coordinate

`y`- the y coordinate

`width`- the width of the rectangle

`height`- the height of the rectangle

See Also:

`drawRect` [\(II-§1.20.20\)](#)

`clearRect` [\(II-§1.20.2\).](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.fillRect

public abstract void

fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

Fills an outlined rounded corner rectangle with this graphics context's current color. The left and right edges are at x and x + width -1 respectively. The top and bottom edges are at y and y + height -1 respectively.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

arcWidth- the horizontal diameter of the arc at the four corners

arcHeight- the vertical diameter of the arc at the four corners

Draws a rounded rectangle filled in with the current color.

See Also:

drawRoundRect [\(II-§1.20.21\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.finalize

public void finalize()

The finalize method ensures that this graphics context's dispose method (II-§1.20.7) is called when this graphics context is no longer referenced.

Overrides:

finalize in class Object (I-§1.12.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.getClipRect

public abstract Rectangle getClipRect()

Returns:

the bounding rectangle of this graphics context's clipping area.

See Also:

clipRect [\(II-§1.20.3\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.getColor

public abstract Color getColor()

Returns:

this graphics context's current color.

See Also:

setColor [\(II-§1.20.36\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.getFont

public abstract Font getFont()

Returns:

this graphics context's current font.

See Also:

setFont [\(II-§1.20.37\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.getFontMetrics

public FontMetrics getFontMetrics()

Returns:

this font metrics of the graphics context's current font.

See Also:

getFont (II-§1.20.33).

public abstract FontMetrics getFontMetrics(Font f)

Parameters:

f – the specified font

Returns:

the font metrics for the specified font.

See Also:

getFont (II-§1.20.33).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.setColor

public abstract void setColor(Color c)

Sets this graphics context's current color to the specified color. All subsequent graphics operations using this graphics context use this specified color.

Parameters:

c- the color

See Also:

Color [\(II-§1.9\)](#)

getColor [\(II-§1.20.32\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.setFont

public abstract void setFont(Font font)

Sets this graphics context's font to the specified font.

All subsequent text operations (such as drawString [\(II-§1.20.22\)](#), drawBytes [\(II-§1.20.10\)](#), and drawChars [\(II-§1.20.11\)](#)) using this graphics context use this font.

Parameters:

font- the font

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.setPaintMode

public abstract void setPaintMode()

Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.setXORMode

public abstract void setXORMode(Color c1)

Sets the paint mode of this graphics context to alternate between this graphics context's current color and the new specified color.

When drawing operations are performed, pixels which are the current color are changed to the specified color and vice versa.

Pixels that are of colors other than those two colors are changed in an unpredictable, but reversible manner; if the same figure is drawn twice then all pixels are restored to their original values.

Parameters:

c1- the second color

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.toString

public String toString()

Returns:

a string representation of this graphics context.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Graphics.translate

public abstract void translate(int x, int y)

Modifies this graphics context so that its new origin corresponds to the point (x, y) in this graphics context's original coordinate system.

Parameters:

x- the x coordinate

y- the y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the background color of offscreen images is white. In Java 1.1, the background color of offscreen images may be system dependent. Applications should use setColor (II-§1.20.36) followed by fillRect (II-§1.20.28) to ensure that an offscreen image is cleared to a specific color.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.21 Class GridBagConstraints

```
public class java.awt.GridBagConstraints
    extends java.lang.Object (I-§1.12)
    implements java.lang.Cloneable (I-§1.22)
{
    // Fields
    public int anchor; §1.21.1
    public int fill; §1.21.2
    public int gridheight; §1.21.3
    public int gridwidth; §1.21.4
    public int gridx; §1.21.5
    public int gridy; §1.21.6
    public Insets insets; §1.21.7
    public int ipadx; §1.21.8
    public int ipady; §1.21.9
    public double weightx; §1.21.10
    public double weighty; §1.21.11

    // the anchor field has one of the following values
    public final static int CENTER; §1.21.12
    public final static int EAST; §1.21.13
    public final static int NORTH; §1.21.14
    public final static int NORTHEAST; §1.21.15
    public final static int NORTHWEST; §1.21.16
    public final static int SOUTH; §1.21.17
    public final static int SOUTHEAST; §1.21.18
    public final static int SOUTHWEST; §1.21.19
    public final static int WEST; §1.21.20

    // the fill field has one of the following values
    public final static int BOTH; §1.21.21
    public final static int HORIZONTAL; §1.21.22
    public final static int NONE; §1.21.23
    public final static int VERTICAL; §1.21.24

    // default value for gridheight, gridwidth
    public final static int REMAINDER; §1.21.25

    // default value for gridx, gridy
    public final static int RELATIVE; §1.21.26

    // Constructors
    public GridBagConstraints(); §1.21.27

    // Methods
    public Object clone(); §1.21.28
```



```
}
```

The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class (II-§1.22).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.anchor

public int anchor

This field is used when the component is smaller than its display area. It determines where, within the area, to place the component. Valid values are:

```
GridBagConstraints.CENTER  
GridBagConstraints.NORTH  
GridBagConstraints.NORTHEAST  
GridBagConstraints.EAST  
GridBagConstraints.SOUTHEAST  
GridBagConstraints.SOUTH  
GridBagConstraints.SOUTHWEST  
GridBagConstraints.WEST  
GridBagConstraints.NORTHWEST
```

The default value is GridBagConstraints.CENTER.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.fill

public int fill

This field is used when the component's display area is larger than the component's requested size. It determines whether to resize the component, and if so, how.

Valid values are:

- GridBagConstraints.NONE: Do not resize the component.
- GridBagConstraints.HORIZONTAL: Make the component wide enough to fill its display area horizontally, but don't change its height.
- GridBagConstraints.VERTICAL: Make the component tall enough to fill its display area vertically, but don't change its width.
- GridBagConstraints.BOTH: Make the component fill its display area entirely.

The default value is GridBagConstraints.NONE.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.gridheight

public int gridheight

Specifies the number of cells in a column for the component's display area.

Use GridBagConstraints.REMAINDER to specify that the component be the last one in its column. Use GridBagConstraints.RELATIVE to specify that the component be the next to last one in its column.

The default value is 1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.gridwidth

public int gridwidth

Specifies the number of cells in a row for the the component's display area.

Use GridBagConstraints.REMAINDER to specify that the component be the last one in its row.
Use GridBagConstraints.RELATIVE to specify that the component be the next to last one in its row.

The default value is 1.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.gridx

public int **gridx**

Specifies the cell at the left of the component's display area, where the leftmost cell has gridx = 0. The value GridBagConstraints.RELATIVE specifies that the component be placed just to the right of the component that was added to the container just before this component was added.

The default value is GridBagConstraints.Relative.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.gridy

public int **gridy**

Specifies the cell at the top of the component's display area, where the topmost cell has gridy = 0. The value GridBagConstraints.RELATIVE specifies that the component be placed just below the component that was added to the container just before this component was added.

The default value is GridBagConstraints.Relative.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.insets

public Insets insets

This field specifies the external padding of the component, the minimum amount of space between the component and the edges of its display area.

The default value is new Insets(0, 0, 0, 0).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.ipadx

public int ipadx

This field specifies the internal padding, that is how much space to add to the minimum size of the component. The width of the component is at least its minimum width plus ipadx*2 pixels.

The default value is 0.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.ipady

public int ipady

This field specifies the internal padding, how much to add to the minimum size of the component. The height of the component is at least its minimum height plus ipadx*2 pixels.

The default value is 0.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.weightx

public double **weightx**

This field specifies how to distribute extra vertical space.

The grid bag layout manager calculates the weight of a row to be the maximum weightx of all the components in a row. If the resulting layout is smaller vertically than the area it needs to fill, the extra space is distributed to each row in proportion to its weight. A row that has weight 0 receives no extra space.

If all the weights are zero, all the extra space appears between the grids of the cell and the top and bottom edges.

The default value of this field is zero.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.weighty

public double **weighty**

This field specifies how to distribute extra horizontal space.

The grid bag layout manager calculates the weight of a column to be the maximum weighty of all the components in a row. If the resulting layout is smaller horizontally than the area it needs to fill, the extra space is distributed to each column in proportion to its weight. A column that has weight 0 receives no extra space.

If all the weights are zero, all the extra space appears between the grids of the cell and the right and left edges.

The default value of this field is zero.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.CENTER

```
public final static int CENTER = 10
```

Puts the component in the center of its display area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.EAST

```
public final static int EAST = 13
```

Puts the component on the right side of its display area, centered vertically.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.NORTH

```
public final static int NORTH = 11
```

Puts the component in the top of its display area, centered horizontally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.NORTHWEST

```
public final static int NORTHWEST = 12
```

Puts the component in the top right corner of its display area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.NORTHWEST

```
public final static int NORTHWEST = 18
```

Puts the component in the top left corner of its display area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.SOUTH

```
public final static int SOUTH = 15
```

Puts the component in the bottom of its display area, centered horizontally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.SOUTHEAST

```
public final static int SOUTHEAST = 14
```

Puts the component in the bottom right corner of its display area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.SOUTHWEST

```
public final static int SOUTHWEST = 16
```

Puts the component in the bottom left corner of its display area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.WEST

```
public final static int WEST = 17
```

Puts the component on the left side of its display area, centered vertically.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.BOTH

```
public final static int BOTH = 1
```

Resizes the component both horizontally and vertically.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.HORIZONTAL

```
public final static int HORIZONTAL = 2
```

Resizes the component horizontally but not vertically.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.NONE

```
public final static int NONE = 0
```

Do not resize the component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.VERTICAL

```
public final static int VERTICAL = 3
```

Resizes the component vertically but not horizontally.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.REMAINDER

```
public final static int REMAINDER = 0
```

Specifies that the component is the last component in its column or row.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagConstraints.RELATIVE

public final static int **RELATIVE** = -1

Specifies that this component is the next to last component in its column or row (gridwidth, gridheight), or that this component be placed next to the previously added component (gridx, gridy).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.GridBagConstraints

public GridBagConstraints()

Creates a grid bag constraint object with all the fields set to their default value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagConstraints.clone

public `Object clone()`

Returns:

a copy of this grid bag constraint.

Overrides:

clone in class `Object` ([I-§1.12.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.22 Class GridBagLayout

```
public class java.awt.GridBagLayout
    extends java.lang.Object (I-§1.12)
    implements java.awt.LayoutManager (II-§1.43)
{
    // Fields
    protected final static int MAXGRIDSIZE; §1.22.1
    protected final static int MINSIZE; §1.22.2

    // Constructors
    public GridBagLayout(); §1.22.3

    // Methods
    public void addLayoutComponent(String name, §1.22.4
                                   Component comp);
    public GridBagConstraints getConstraints(Component comp); §1.22.5
    public void layoutContainer(Container target); §1.22.6
    protected GridBagConstraints §1.22.7
        lookupConstraints(Component comp);
    public Dimension minimumLayoutSize(Container target); §1.22.8
    public Dimension preferredLayoutSize(Container target); §1.22.9
    public void removeLayoutComponent(Component comp); §1.22.10
    public void §1.22.11
        setConstraints(Component comp, GridBagConstraints constraints);
    public String toString(); §1.22.12
}
```

The grid bag layout manager is a flexible layout manager that aligns components horizontally and vertically, without requiring that the components be the same size.

Each grid bag layout manager uses a rectangular grid of cells, with each component occupying one or more cells (called its *display area*). Each component in a grid bag layout is associated with a set of constraints contained within a GridBagConstraints instance that specifies how the component is to be laid out within its display area.

The manner in which the grid bag layout manager places a set of components depends on each component's constraints and its minimum size, as well as the preferred size of the components' container.

To use a grid bag layout effectively, one or more components must have a customized GridBagConstraints objects created for it.

The fields of the GridBagConstraints object are described more fully in §1.21.

The following figure shows ten components (all buttons) managed by a grid bag layout manager:

```
{ewc msdncd, EWGraphic, AWT0es 0 /a "sunref.BMP"}
```


Each of the ten components has the fill field of their constraint set to GridBagConstraints.BOTH. In addition, the buttons have the following non-default constraints:

- Button1, Button2, Button3: weightx=1.0
- Button4: weightx=1.0, gridwidth=GridBagConstraints.REMAINDER
- Button5: gridwidth=GridBagConstraints.REMAINDER
- Button6: gridwidth=GridBagConstraints.RELATIVE
- Button7: gridwidth=GridBagConstraints.REMAINDER
- Button8: grid, weighty=1.0,
- Button9, Button 10: gridwidth=GridBagConstraints.REMAINDER

The following code implements the example shown above:

```
import java.awt.*;
import java.util.*;
import java.applet.Applet;
public class GridBagEx1 extends Applet {
    protected void makebutton(String name,
                               GridBagLayout gridbag,
                               GridBagConstraints c) {
        Button button = new Button(name);
        gridbag.setConstraints(button, c);
        add(button);
    }

    public void init() {
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setFont(new Font("Helvetica", Font.PLAIN, 14));
        setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton("Button1", gridbag, c);
        makebutton("Button2", gridbag, c);
        makebutton("Button3", gridbag, c);

        // end row
        c.gridwidth = GridBagConstraints.REMAINDER;
        makebutton("Button4", gridbag, c);
        c.weightx = 0.0; //reset to the default
        makebutton("Button5", gridbag, c); //another row

        // next-to last in row
        c.gridwidth = GridBagConstraints.RELATIVE;
        makebutton("Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        makebutton("Button7", gridbag, c);
        c.gridwidth = 1; // reset to the default
        c.gridheight = 2;
        c.weighty = 1.0;
        makebutton("Button8", gridbag, c);
```



```

        c.weighty = 0.0;                //reset to the default

    // end row
    c.gridwidth = GridBagConstraints.REMAINDER;
    c.gridheight = 1;                  // reset to the default
    makebutton("Button9", gridbag, c);
    makebutton("Button10", gridbag, c);
    resize(300, 100);
}

public static void main(String args[]) {
    Frame f = new Frame("GridBag Layout Example");
    GridBagEx1 ex1 = new GridBagEx1();
    ex1.init();
    f.add("Center", ex1);
    f.pack();
    f.show();
}
}

```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagLayout.MAXGRIDSIZE

protected final static int MAXGRIDSIZE = 128

The maximum number of grid positions (both horizontally and vertically) that can be laid out by the grid bag layout.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.MINSIZE

```
protected final static int MINSIZE = 1
```

The smallest grid that can be laid out by the grid bag layout.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagLayout.GridBagLayout

public GridBagLayout ()

Creates a grid bag layout manager.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.addLayoutComponent

public void addLayoutComponent(String name, Component comp)

This method is not used by the grid bag layout manager.

Parameters:

name- a tag understood by the layout manager

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.getConstraints

public GridBagConstraints getConstraints(Component comp)

Parameters:

comp- the component to be queried

Returns:

the constraint for the specified component in this grid bag layout. A copy of the constraint object is returned.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.layoutContainer

public void layoutContainer(Container target)

Lays out the container argument using this grid bag layout.

This method reshapes the components in the specified container in order to satisfy the constraints of this GridBagLayout object.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target– the container in which to do the layout

See Also:

Container (II-§1.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.lookupConstraints

protected GridBagConstraints

lookupConstraints(Component comp)

Retrieves the constraints for the specified component in this grid bag layout. The return value is not a copy, but is the actual GridBagConstraints object used by the layout mechanism.

Parameters:

comp- the component to be queried

Returns:

the constraints for the specified component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.minimumLayoutSize

public **Dimension** **minimumLayoutSize**(**Container** **target**)

Determines the minimum size of the **target** container using this grid bag layout.

This **method** is called when a container calls its layout **method** ([II-§1.11.11](#)). Most applications do not call this **method** directly.

Parameters:

target– the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

preferredLayoutSize ([II-§1.22.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.preferredLayoutSize

public Dimension preferredLayoutSize(Container target)

Determines the preferred size of the target container using this grid bag layout.

Most applications do not call this method directly. This method is called when a container calls its preferredSize method (II-§1.11.17).

Parameters:

target– the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

minimumLayoutSize (II-§1.22.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.removeLayoutComponent

public void removeLayoutComponent(Component comp)

Removes the specified component from this layout.

Most applications do not call this method directly. This method is called when a container calls its remove [\(II-§1.11.19\)](#) or removeAll [\(II-§1.11.20\)](#) methods.

Parameters:

comp- the component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridBagLayout.setConstraints

```
public void setConstraints(Component comp, GridBagConstraints constraints)
```

Sets the constraints for the specified component in this layout.

Parameters:

`comp`- the component to be modified

`constraints`- the constraints to be applied

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridBagLayout.toString

public String toString()

Returns:

a string representation of this grid bag layout.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.23 Class GridLayout

```
public class java.awt.GridLayout
    extends java.lang.Object (I-§1.12)
    implements java.awt.LayoutManager (II-§1.43)
{
    // Constructors
    public GridLayout(int rows, int cols); §1.23.1
    public GridLayout(int rows, int cols, §1.23.2
                      int hgap, int vgap);
    // Methods
    public void addLayoutComponent(String name, §1.23.3
                                   Component comp);
    public void layoutContainer(Container target); §1.23.4
    public Dimension minimumLayoutSize(Container target); §1.23.5
    public Dimension preferredLayoutSize(Container target); §1.23.6
    public void removeLayoutComponent(Component comp); §1.23.7
    public String toString(); §1.23.8
}
```

This grid layout manager causes the container's components to be laid out in a rectangular grid. The container is split into equal-sized rectangles: one component is placed into each rectangle.

For example, the following code says to lay out the six buttons into three rows and two columns:

```
import java.awt.*;
import java.applet.Applet;
public class buttonGrid extends Applet {
    public void init() {
        setLayout(new GridLayout(3,2));
        add(new Button("1"));
        add(new Button("2"));
        add(new Button("3"));
        add(new Button("4"));
        add(new Button("5"));
        add(new Button("6"));
    }
}
```

It produces the following output{ewc msdncl, EWGraphic, AWT0et 0 /a "sunref.BMP"}:

{ewl msdncl.dll, ewcright, /c"Microsoft"}

GridLayout.GridLayout

public GridLayout(int rows, int cols)

Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size.

One, but not both of rows and columns can be zero, which means that any number of objects can be placed in a row or in a column.

Parameters:

rows- the rows; zero means "any number"

cols- the columns; zero means "any number"

public GridLayout(int rows, int cols, int hgap, int vgap)

Creates a grid layout with the specified number of rows and columns. All components in the layout are given equal size.

In addition the horizontal and vertical gaps are set to the specified values. The horizontal gaps are placed at the left and right edge, and between each of the columns. The vertical gaps are placed at the top and bottom edge, and between each of the rows.

One, but not both of rows and columns can be zero, which means that any number of objects can be placed in a row or in a column.

Parameters:

rows- the rows; zero means "any number"

cols- the columns; zero means "any number"

hgap- the horizontal gap

vgap- the vertical gap

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.addComponent

public void addLayoutComponent(String name, Component comp)

This method is not used by the grid layout manager.

Parameters:

name- a tag

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.layoutContainer

public void layoutContainer(Container target)

Lays out the container argument using this layout.

This method reshapes the components in the specified target container in order to satisfy the constraints of the GridLayout object.

The grid layout manager determines the size of individual components by dividing the free space in the container into equal-sized portions according to the number of rows and columns in the layout. The container's free space equals the container's size minus any insets and any specified horizontal or vertical gap. All components in a grid layout are given the same size

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

target - the container in which to do the layout

See Also:

Container (II-§1.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.minimumLayoutSize

public Dimension minimumLayoutSize(Container target)

Determines the minimum size of the container argument using this grid layout.

The minimum width of a grid layout is the largest minimum width of any of the widths in the container times the number of columns, plus the horizontal padding times the number of columns plus 1, plus the left and right insets of the target container.

The minimum height of a grid layout is the largest minimum height of any of the widths in the container times the number of rows, plus the vertical padding times the number of rows plus 1, plus the top and left insets of the target container.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

`target`– the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

`preferredLayoutSize` (II-§1.23.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.preferredLayoutSize

public Dimension preferredLayoutSize(Container target)

Determines the preferred size of the container argument using this grid layout.

The preferred width of a grid layout is the largest preferred width of any of the widths in the container times the number of columns, plus the horizontal padding times the number of columns plus 1, plus the left and right insets of the target container.

The preferred height of a grid layout is the largest preferred height of any of the widths in the container times the number of rows, plus the vertical padding times the number of rows plus 1, plus the top and left insets of the target container.

Most applications do not call this method directly. This method is called when a container calls its preferredSize method (II-§1.11.17).

Parameters:

`target`– the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

`minimumLayoutSize` (II-§1.23.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.removeLayoutComponent

public void removeLayoutComponent(Component comp)

This method is not used by the grid layout manager.

Parameters:

comp- the component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

GridLayout.toString

public String toString()

Returns:

a string representation of this grid layout.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.24 Class Image

```
public abstract class java.awt.Image
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public final static Object UndefinedProperty; §1.24.1

    // Constructors
    public Image(); §1.24.2

    // Methods
    public abstract void flush(); §1.24.3
    public abstract Graphics getGraphics(); §1.24.4
    public abstract int getHeight(ImageObserver observer); §1.24.5
    public abstract Object §1.24.6
        getProperty(String name, ImageObserver observer);
    public abstract ImageProducer getSource(); §1.24.7
    public abstract int getWidth(ImageObserver observer) §1.24.8
}
```

The abstract class Image is the superclass of all classes that represent graphical images.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Image.UndefinedProperty

```
public final static Object UndefinedProperty = new Object()
```

The UndefinedProperty object should be returned whenever a property which was not defined for a particular image is fetched.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Image.Image

public Image ()

The default constructor for an image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.flush

public **abstract void flush()**

Flushes all resources being used by this Image object.

These resources include any pixel data that is being cached for rendering to the screen, as well as any system resources that are being used to store data or pixels for the image.

The Image object is reset to a state similar to when it was first created so that if it is again rendered, the image data must be recreated or fetched again from its source.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.getGraphics

public abstract Graphics getGraphics()

Creates a graphics context (II-§1.20) for drawing to an off-screen image. This method can only be called for off-screen images, which are created with the createImage method (II-§1.10.7) with two integer arguments.

Returns:

a graphics context to draw to the off-screen image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.getHeight

public abstract int getHeight(ImageObserver observer)

Determines the height of this image. If the height is not yet known, the observer is notified later.

Parameters:

observer- an object waiting for the image to be loaded

Returns:

the height of the image, or -1 if the height is not yet known.

See Also:

getWidth [\(II-§1.24.8\)](#)
ImageObserver [\(II-§2.11\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.getProperty

public abstract Object

getProperty(String name, ImageObserver observer)

Gets a property of this image by name.

Individual property names are defined by the various image formats. If a property is not defined for a particular image, this method returns the UndefinedProperty object.

If the properties for this image are not yet known, this method returns null and the ImageObserver object is notified later.

The property name "comment" should be used to store an optional comment which can be presented to the application as a description of the image, its source, or its author.

Parameters:

name- a property name

observer- an object waiting for this image to be loaded

Returns:

the value of the named property.

See Also:

ImageObserver (II-§2.11)

UndefinedProperty (II-§1.24.1).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.getSource

public abstract ImageProducer getSource()

Returns:

the image producer (II-§2.12) that produces the pixels for this image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Image.getWidth

public abstract int getWidth(ImageObserver observer)

Determines the width of this image. If the width is not yet known, the observer is notified later.

Parameters:

observer- an object waiting for the image to be loaded

Returns:

the width of this image, or -1 if the width is not yet known.

See Also:

getHeight (II-§1.24.5)
ImageObserver (II-§2.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.25 Class Insets

```
public class java.awt.Insets
    extends java.lang.Object (I-§1.12)
    implements java.lang.Cloneable (I-§1.22)
{
    // Fields
    public int bottom; §1.25.1
    public int left; §1.25.2
    public int right; §1.25.3
    public int top; §1.25.4

    // Constructors
    public Insets(int top, int left, int bottom, int right); §1.25.5

    // Methods
    public Object clone(); §1.25.6
    public String toString(); §1.25.7
}
```

The Insets object is a representation of the borders of a container. It specifies the space that a container must leave at each of its edges. The space can either be a border, blank space, or a title.

See Also:

LayoutManager (II-§1.43)
Container (II-§1.11).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Insets.bottom

public int bottom

The inset from the bottom.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.left

public int left

The inset from the left.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.right

public int right

The inset from the right.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.top

public int top

The inset from the top.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.Insets

public Insets(int top, int left, int bottom, int right)

Creates and initializes a new Inset with the specified top, left, bottom, and right insets.

Parameters:

top- the inset from the top

left- the inset from the left

bottom- the inset from the bottom

right- the inset from the right

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.clone

public `Object clone()`

Returns:

a copy of this inset.

Overrides:

clone in class `Object` ([I-§1.12.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Insets.toString

public String toString()

Returns:

a string representation of this inset.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.26 Class Label

```
public class java.awt.Label
    extends java.awt.Component (II-§1.10)
{
    // Fields
    public final static int CENTER; §1.26.1
    public final static int LEFT; §1.26.2
    public final static int RIGHT; §1.26.3

    // Constructors
    public Label(); §1.26.4
    public Label(String label); §1.26.5
    public Label(String label, int alignment); §1.26.6

    // Methods
    public void addNotify(); §1.26.7
    public int getAlignment(); §1.26.8
    public String getText(); §1.26.9
    protected String  paramString(); §1.26.10
    public void setAlignment(int alignment); §1.26.11
    public void setText(String label); §1.26.12
}
```

A label is a component for placing text in a container. The text can be changed by the application, but a user cannot edit it directly.1

For example, the code:

```
setLayout(new FlowLayout(FlowLayout.CENTER,10,10));
add(new Label("Hi There!"));
add(new Label("Another Label"));
```

produces the following:

```
{ewc msdncd, EWGraphic, AWT0ew 0 /a "sunref.BMP"}
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Label.CENTER

```
public final static int CENTER
```

Indicates that the label should be centered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Label.LEFT

public final static int LEFT

Indicates that the label should be left justified.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.RIGHT

```
public final static int RIGHT
```

Indicates that the label should be right justified.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Label.Label

`public Label()`

Constructs an empty label with whose text is left justified.

`public Label(String label)`

Constructs a new label with the specified string of text left justified.

Parameters:

`label`- the text that makes up the label

`public Label(String label, int alignment)`

Constructs a new label with the specified string of text and the specified alignment.

The alignment value must be one of Label.LEFT, Label.RIGHT, or Label.CENTER.

Parameters:

`label`- the string that makes up the label

`alignment`- the alignment value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.addNotify

public void addNotify()

This method calls the createLabel method (II-§1.41.12) of this object's toolkit (II-§1.10.20) in order to create a LabelPeer (II-§3.11) for this label. This peer allows the application to change the look of a label without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.getAlignment

public int getAlignment()

Returns:

the current alignment of this label.

See Also:

setAlignment (II-§1.26.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.getText

public String **getText**()

Returns:

the text of this label.

See Also:

setText (II-§1.26.12).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label paramString

protected String paramString()

Returns the parameter string representing the state of this label. This string is useful for debugging.

Returns:

the parameter string of this label.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.setAlignment

public void **setAlignment**(int alignment)

Sets the alignment for this label to the specified alignment.

Parameters:

alignment- the alignment value

Throws

IllegalArgumentException (I-§1.32)

If an improper alignment was given.

See Also:

getAlignment (II-§1.26.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Label.setText

public void setText(String label)

Sets the text for this label to the specified text.

Parameters:

label - the text that makes up the label

See Also:

getText (II-§1.26.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse, keyboard, or focus events to a label. In Java 1.1, the AWT sends to the label all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.27 Class List

```
public class java.awt.List
    extends java.awt.Component (II-§1.10)
{
    // Constructors
    public List(); §1.27.1
    public List(int rows, boolean multipleSelections); §1.27.2

    // Methods
    public void addItem(String item); §1.27.3
    public void addItem(String item, int index); §1.27.4
    public void addNotify(); §1.27.5
    public boolean allowsMultipleSelections(); §1.27.6
    public void clear(); §1.27.7
    public int countItems(); §1.27.8
    public void delItem(int position); §1.27.9
    public void delItems(int start, int end); §1.27.10
    public void deselect(int index); §1.27.11
    public String getItem(int index); §1.27.12
    public int getRows(); §1.27.13
    public int getSelectedIndex(); §1.27.14
    public int[] getSelectedIndexes(); §1.27.15
    public String getSelectedItem(); §1.27.16
    public String[] getSelectedItems(); §1.27.17
    public int getVisibleIndex(); §1.27.18
    public boolean isSelected(int index); §1.27.19
    public void makeVisible(int index); §1.27.20
    public Dimension minimumSize(); §1.27.21
    public Dimension minimumSize(int rows); §1.27.22
    protected String  paramString(); §1.27.23
    public Dimension preferredSize(); §1.27.24
    public Dimension preferredSize(int rows); §1.27.25
    public void removeNotify(); §1.27.26
    public void replaceItem(String newValue, int index); §1.27.27
    public void select(int index); §1.27.28
    public void setMultipleSelections(boolean v); §1.27.29
}
```

The List component presents the user with a scrolling list of text items. The list can be set up either so that the user can pick one item or to pick multiple items.

For example, the code:

```
List l = new List(4, false);
l.addItem("Mercury");
l.addItem("Venus");
```



```

l.addItem("Earth");
l.addItem("JavaSoft");
l.addItem("Mars");
l.addItem("Jupiter");
l.addItem("Saturn");
l.addItem("Uranus");
l.addItem("Neptune");
l.addItem("Pluto");
add(l);

```

produces the following scrolling list:

```
{ewc msdncd, EWGraphic, AWT0ex 0 /a "sunref.BMP"}
```

Clicking¹ on an item that isn't selected selects it. Clicking on an item that is already selected deselects it. In the above example, since the second argument when creating the new scrolling list is false, only one item can be selected at a time from the scrolling list. Selecting any item causes any other selected item to be automatically deselected.

When an item is clicked and becomes selected, AWT sends a list select event (II-§1.14.18) to the scrolling list. When an item is clicked and becomes deselected, AWT sends a list deselect event (II-§1.14.17) to the scrolling list. The event's target is the scrolling list, and its object is an Integer (I-§1.8) giving the index of the item in the list.

When the user double clicks on an item in a scrolling list, AWT sends an action event (II-§1.14.11) to the scrolling list after the list select or deselect event. The event's target is the scrolling list, and its object is the string label of the item selected or deselected.

When the user selects return inside a scrolling list, AWT also sends an action event (II-§1.14.11) to the scrolling list. The event's target is the scrolling list, and its object is the string label of the last item selected or deselected in the scrolling list.

If an application wants to perform some action based on an item being selected or deselected, it must override the handleEvent method of the scrolling list or of one of its containing windows. The code to perform that should be of the form:

```

public boolean handleEvent(Event event) {
    switch(event.id) {
        case Event.LIST_SELECT:
            <do something if event.target is scrolling list>
        case Event.LIST_DESELECT:
            <do something if event.target is scrolling list>
        default:
            return super.handleEvent(event) ;
    }
}

```

If the application wants to perform some action based on the user double clicking or hitting the return key, it should override the action method (II-§1.10.1) of the scrolling list or of one of its

containing windows. Alternatively, it can override the `handleEvent` method as above and check to see if the event's id field is `Event.ACTION_EVENT`.

For multiple-selection scrolling lists, it is considered a better user interface to use an external event (such as clicking on a button) to trigger the action.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


List.List

public List()

Creates a new scrolling list. Initially there are no visible lines, and only one item can be selected from the list.

public List(int rows, boolean multipleSelections)

Creates a new scrolling list initialized to display the specified number of rows. If the multipleSelections argument is true, then the user can select multiple items from the list. If it is false, only one item at a time can be selected.

Parameters:

rows- the number of items to show

multipleSelections- if true then multiple selections are allowed; otherwise, only one item can be selected at a time

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.AddItem

public void AddItem(String item)

Adds the specified string to the end of this scrolling list.

Parameters:

item- the string to be added

public void AddItem(String item, int index)

Adds the specified string to this scrolling list at the specified position.

The index argument is 0-based. If the index is -1, or greater than or equal to the number of items already in the list, then the item is added at the end of the list.

Parameters:

item- the string to be added

index- the position at which to put in the item

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.addNotify

public void addNotify()

This method calls the createList method ([II-§1.41.13](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a ListPeer ([II-§3.12](#)) for this scrolling list. This peer allows the application to change the look of a scrolling list without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component ([II-§1.10.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.allowsMultipleSelections

public boolean allowsMultipleSelections()

Returns:

true if this scrolling list allows multiple selections; false otherwise.

See Also:

setMultipleSelections (II-§1.27.29).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.clear

public void clear()

Removes all items from this scrolling list.

See Also:

delItem [\(II-§1.27.9\)](#)

delItems [\(II-§1.27.10\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.countItems

public int countItems()

Returns:

the number of items in this list.

See Also:

getItem [\(II-§1.27.12\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.delItem

public void delItem(int position)

Deletes the item at the specified position from this scrolling list.

Parameters:

position- the index of the item to delete

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


List.delItems

public void delItems(int start, int end)

Deletes the items in the range start is less than or equal to item is less than or equal to end from this scrolling list.

Parameters:

start- the index of the first element to delete

end- the index of the last element to delete

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.deselect

public void deselect(int index)

Deselects the item at the specified index of this scrolling list.

If the item at the specified index is not selected, or if the index is out of range, then the operation is ignored.

Parameters:

index- the position of the item to deselect

See Also:

select ([II-§1.27.28](#))

getSelectedItem ([II-§1.27.16](#))

isSelected ([II-§1.27.19](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.GetItem

public String getItem(int index)

Parameters:

index- the position of the item

Returns:

the string of this scrolling list at the specified index.

See Also:

countItems (II-§1.27.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.getRows

```
public int getRows ()
```

Returns:

the number of visible lines in this scrolling list.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


List.getSelectedIndex

public int `getSelectedIndex()`

Returns:

the index of the selected item on this scrolling list, or -1 if no items are selected or more than one item is selected.

See Also:

select (II-§1.27.28)
deselect (II-§1.27.11)
isSelected (II-§1.27.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.getSelectedIndexes

public int[] `getSelectedIndexes()`

Returns:

an array of the selected indexes of this scrolling list.

See Also:

select (II-§1.27.28)
deselect (II-§1.27.11)
isSelected (II-§1.27.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.SelectedItem

public String SelectedItem()

Returns:

the selected item on this scrolling list, or null if either no items are selected or more than one item is selected.

See Also:

select (II-§1.27.28)
deselect (II-§1.27.11)
isSelected (II-§1.27.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.getSelectedItems

```
public String[] getSelectedItems()
```

Returns:

an array of the selected items on this scrolling list.

See Also:

select [\(II-§1.27.28\)](#)
deselect [\(II-§1.27.11\)](#)
isSelected [\(II-§1.27.19\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


List.VisibleIndex

public int **getVisibleIndex**()

Returns:

the index of the item in this scrolling list that was last made visible by the makeVisible method (II-§1.27.20).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.isSelected

`public boolean isSelected(int index)`

Determines if a specified item in this scrolling list is selected. No error occurs if the index argument is less than 0 or greater than or equal to the number of items in this scrolling list.

Parameters:

`index`– the item to be checked

Returns:

true if the item at the specified index has been selected; false otherwise.

See Also:

select [\(II-§1.27.28\)](#)
deselect [\(II-§1.27.11\)](#)
isSelected [\(II-§1.27.19\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.makeVisible

public void makeVisible(int index)

Forces the item at the specified index in this scrolling list to be visible.

No error occurs if the index argument is less than 0 or greater than or equal to the number of items in this scrolling list.

Parameters:

index- the position of the item

See Also:

setVisibleIndex (II-§1.27.18).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.minimumSize

public **Dimension** **minimumSize()**

Determines the minimum size of this scrolling list. If the application has specified the number of visible rows, and that number is greater than 0, the peer's minimumSize method (II-§3.12.7) is called with the number of rows in order to determine the minimum size.

If this scrolling list does not have a peer, or if the number of visible rows is less than or equal to zero, the superclass's minimumSize method (II-§1.10.40) is called to determine the minimum size.

Returns:

the minimum dimensions needed to display this scrolling list.

Overrides:

minimumSize in class Component (II-§1.10.40).

public **Dimension** **minimumSize(int rows)**

Determines the minimum size of a scrolling list with the specified number of rows. This scrolling list's peer's minimumSize method (II-§3.12.7) is called with the number of rows in order to determine the minimum size.

If this scrolling list does not have a peer the superclass's minimumSize method (II-§1.10.40) is called to determine the minimum size.

Parameters:

rows— number of rows

Returns:

the minimum dimensions needed to display the specified number of rows in a scrolling list.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List paramString

protected String paramString()

Returns the parameter string representing the state of this scrolling list. This string is useful for debugging.

Returns:

the parameter string of this scrolling list.

Overrides:

paramString in class Component (II-§1.10.51).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.preferredSize

public Dimension preferredSize()

Determines the preferred size of this scrolling list. If the application has specified the number of visible rows, and that number is greater than 0, the peer's preferredSize method (II-§3.12.8) is called with the number of rows in order to determine the preferred size.

If this scrolling list does not have a peer, or if the number of visible rows is less than or equal to zero, the superclass's preferredSize method (II-§1.10.40) is called to determine the preferred size.

Returns:

the preferred dimensions for displaying this scrolling list.

Overrides:

preferredSize in class Component (II-§1.10.53).

public Dimension preferredSize(int rowsn)

Determines the preferred size of a scrolling list with the specified number of rows. This scrolling list's peer's preferredSize method (II-§3.12.8) is called with the number of rows in order to determine the preferred size.

If this scrolling list does not have a peer the superclass's preferredSize method (II-§1.10.40) is called to determine the preferred size.

Parameters:

`rowsn` - number of rows

Returns:

the preferred dimensions for displaying the specified number of rows.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.removeNotify

public void removeNotify()

Notifies this scrolling list to destroy its peer.

Overrides:

removeNotify in class Component (II-§1.10.58).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.replaceItem

public void replaceItem(String newValue, int index)

Replaces the item at the given index in the scrolling list with the new string.

Parameters:

newValue- the new value

index- the position

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


List.select

public void select(int index)

Selects the item at the specified index in the scrolling list.

Parameters:

index- the position of the item to select

See Also:

getSelectedItem ([II-§1.27.16](#))

deselect ([II-§1.27.11](#))

isSelected ([II-§1.27.19](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

List.setMultipleSelections

public void setMultipleSelections (boolean v)

Sets whether this scrolling list allows multiple selections.

Parameters:

v – if true then multiple selections are allowed; otherwise, only one item can be selected at a time

See Also:

allowsMultipleSelections [\(II-§1.27.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse, keyboard, or focus events to a list. In Java 1.1, the AWT sends to the list all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.28 Class MediaTracker

```
public class java.awt.MediaTracker
    extends java.lang.Object (I-§1.12)
{
    // Fields
    public final static int ABORTED;           §1.28.1
    public final static int COMPLETE;         §1.28.2
    public final static int ERRORED;          §1.28.3
    public final static int LOADING;          §1.28.4

    // Constructors
    public MediaTracker(Component comp); §1.28.5

    // Methods
    public void addImage(Image image, int id); §1.28.6
    public void addImage(Image image, int id, int w, int h); §1.28.7
    public boolean checkAll(); §1.28.8
    public boolean checkAll(boolean load); §1.28.9
    public boolean checkID(int id); §1.28.10
    public boolean checkID(int id, boolean load); §1.28.11
    public Object[] getErrorsAny(); §1.28.12
    public Object[] getErrorsID(int id); §1.28.13
    public boolean isErrorAny(); §1.28.14
    public boolean isErrorID(int id); §1.28.15
    public int statusAll(boolean load); §1.28.16
    public int statusID(int id, boolean load); §1.28.17
    public void waitForAll(); §1.28.18
    public boolean waitForAll(long ms); §1.28.19
    public void waitForID(int id); §1.28.20
    public boolean waitForID(int id, long ms); §1.28.21
}
```

The MediaTracker class is a utility class to trace the status of a number of media objects. Media objects could include images as well as audio clips, though currently only images are supported.

To use the media tracker, create an instance of the MediaTracker class and then call the addImage method (II-§1.28.6) for each image to be tracked. In addition each image can be assigned a unique identifier. The identifier controls both the priority order in which the images are fetched as well as identifying unique subsets of the images that can be waited on independently. Images with a lower ID are loaded in preference to those with a higher ID number.

Here is an example:

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Image;
import java.awt.Graphics;
```



```

import java.awt.MediaTracker;
public class ImageBlaster extends Applet
implements Runnable {
    MediaTracker tracker;
    Image bg;
    Image anim[] = new Image[5];
    int index;
    Thread animator;
    // Get the images for the background (id == 0) and the
    // animation frames (id == 1) and add them to the
    // Media Tracker
    public void init() {
        tracker = new MediaTracker(this);
        bg = getImage(getDocumentBase(),
            "images/background.gif");
        tracker.addImage(bg, 0);
        for (int i = 0; i < 5; i++) {
            anim[i] = getImage(getDocumentBase(),
                "images/anim"+i+".gif");
            tracker.addImage(anim[i], 1);
        }
    }

    // Start the animation thread.
    public void start() {
        animator = new Thread(this);
        animator.start();
    }

    // Stop the animation thread.
    public void stop() {
        animator.stop();
        animator = null;
    }

    // Run the animation thread.
    // First wait for the background image to fully load
    // and point. Then wait for all of the animation
    // frames to finish loading. Finally loop and
    // increment the animation frame index.
    public void run() {
        try {
            tracker.waitForID(0);
            tracker.waitForID(1);
        } catch (InterruptedException e) {
            return;
        }

        Thread me = Thread.currentThread();
        while (animator == me) {
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                break;
            }
        }
        synchronized (this) {
            index++;
        }
    }
}

```



```

        if (index >= anim.length) {
            index = 0;
        }
        repaint();
    }
}

// The background image fills our frame so we don't
// need to clear the applet on repaints; just call the
// paint method
public void update(Graphics g) {
    paint(g);
}

// Paint a large red rectangle if there are any errors
// loading the images. Otherwise, always paint the
// background so that it appears incrementally as it
// is loading. Finally, only paint the current
// animation frame if all of the frames (id == 1)
// are done loading, so that we don't get partial
// animations.
public void paint(Graphics g) {
    if ((tracker.statusAll() &&
        MediaTracker.ERROR) != 0) {
        g.setColor(Color.red);
        g.fillRect(0, 0, size().width, size().height);
        return;
    }
    g.drawImage(bg, 0, 0, this);
    if ((tracker.statusID(1) &&
        MediaTracker.COMPLETE) != 0) {
        g.drawImage(anim[index], 10, 10, this);
    }
}
}

```

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.ABORTED

```
public final static int ABORTED = 2
```

Flag indicating the download of some media was aborted.

See Also:

statusAll [\(II-§1.28.16\)](#)

statusID [\(II-§1.28.17\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MediaTracker.COMPLETE

```
public final static int COMPLETE = 8
```

Flag indicating the download of media completed successfully.

See Also:

statusAll [\(II-§1.28.16\)](#)

statusID [\(II-§1.28.17\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MediaTracker.ERROR

```
public final static int ERROR = 4
```

Flag indicating the download of some media encountered an error.

See Also:

statusAll [\(II-§1.28.16\)](#)

statusID [\(II-§1.28.17\)](#).

```
{ewl msdncl.dll, ewcright, /c"Microsoft"}
```


MediaTracker.LOADING

```
public final static int LOADING = 1
```

Flag indicating some media is currently being loaded.

See Also:

statusAll [\(II-§1.28.16\)](#)

statusID [\(II-§1.28.17\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MediaTracker.MediaTracker

public **MediaTracker**(**Component** comp)

Creates a media tracker to track images for a given component.

Parameters:

comp- the component on which the images will eventually be drawn

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.addImage

public void addImage(Image image, int id)

Adds an image to the list of images being tracked by this media tracker. The image will eventually be rendered at its default (unscaled) size.

Parameters:

image- the image to be tracked

id- the identifier used to later track this image

public void addImage(Image image, int id, int w, int h)

Adds a scaled image to the list of images being tracked by this media tracker. The image will eventually be rendered at the indicated width and height.

Parameters:

image- the image to be tracked

id- the identifier used to later track this image

w- the width at which the image will be rendered

h- the height at which the image will be rendered

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.checkAll

`public boolean checkAll()`

Checks to see if all images being tracked by this media tracker have finished loading.

This method does not start loading the images if they are not already loading.

If there is an error while loading or scaling an image then that image is considered to have finished loading. Use the `isErrorAny` method (II-§1.28.14) or `isErrorID` method (II-§1.28.15) to check for errors.

Returns:

true if all images have finished loading, were aborted, or encountered an error; false otherwise.

See Also:

`checkID` (II-§1.28.10).

`public boolean checkAll(boolean load)`

Checks to see if all images being tracked by this media tracker have finished loading.

If the load flag is true, then starts loading any images that are not yet being loaded.

If there is an error while loading or scaling an image then that image is considered to have finished loading. Use the `isErrorAny` method (II-§1.28.14) or `isErrorID` method (II-§1.28.15) to check for errors.

Parameters:

`load`- if true, starts loading any images that are not yet being loaded

Returns:

true if all images have finished loading, were aborted, or encountered an error; false otherwise.

See Also:

`checkID` (II-§1.28.11).


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MediaTracker.checkID

public boolean checkID(int id)

Checks to see if all images tracked by this media tracker that are tagged with the specified identifier have finished loading

This method does not start loading the images if they are not already loading.

If there is an error while loading or scaling an image then that image is considered to have finished loading. Use the isErrorAny method ([II-§1.28.14](#)) or isErrorID method ([II-§1.28.15](#)) to check for errors.

Parameters:

id- the identifier of the images to check

Returns:

true if all images have finished loading, were aborted, or encountered an error; false otherwise.

See Also:

checkAll ([II-§1.28.8](#)).

public boolean checkID(int id, boolean load)

Checks to see if all images tracked by this media tracker that are tagged with the specified identifier have finished loading.

If the load flag is true, then start, loading any images that are not yet being loaded.

If there is an error while loading or scaling an image then that image is considered to have finished loading. Use the isErrorAny method ([II-§1.28.14](#)) or isErrorID method ([II-§1.28.15](#)) to check for errors.

Parameters:

id- the identifier of the images to check

`load`- if true, starts loading any images that are not yet being loaded

Returns:

true if all images have finished loading, were aborted, or encountered an error; false otherwise.

See Also:

`checkAll` ([II-§1.28.8](#)).

{`ewl msdncd.dll, ewcright, /c"Microsoft"`}

MediaTracker.getErrorsAny

public Object[] getErrorsAny()

Returns:

an array of media objects tracked by this media tracker that have encountered an error, or null if there are none with errors.

See Also:

isErrorAny (II-§1.28.14)

getErrorsID (II-§1.28.13).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.getErrorsID

public Object[] getErrorsID(int id)

Parameters:

id- the identifier of the images to check

Returns:

an array of media objects tracked by this media tracker with the specified identifier that have encountered an error, or null if there are none with errors.

See Also:

isErrorID [\(II-§1.28.15\)](#)

getErrorsAny [\(II-§1.28.12\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.isErrorAny

public boolean isErrorAny()

Returns:

true if any of the images tracked by this media tracker had an error during loading; false otherwise.

See Also:

isErrorID [\(II-§1.28.15\)](#)

getErrorsAny [\(II-§1.28.12\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.isErrorID

public boolean isErrorID(int id)

Checks the error status of all of the images tracked by this media tracker with the specified ID.

Parameters:

id- the identifier of the images to check

Returns:

true if any of the images with the specified identifier had an error during loading; false otherwise.

See Also:

isErrorAny [\(II-§1.28.14\)](#)
getErrorsID [\(II-§1.28.13\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.statusAll

public int statusAll(boolean load)

Calculates and returns the bitwise inclusive OR of the status of all the media being tracked by this media tracker. The possible flags are specified by the following four constants:

- MediaTracker.LOADING (II-§1.28.4)
- MediaTracker.ABORTED (II-§1.28.1)
- MediaTracker.ERRORERD (II-§1.28.3)
- MediaTracker.COMPLETE (II-§1.28.2)

An image that hasn't started loading has zero as its status.

If the load flag is true, then starts loading any images that are not yet being loaded.

Parameters:

load- if true, starts loading any images that are not yet being loaded

Returns:

the bitwise inclusive OR of the status of all of the media being tracked.

See Also:

statusID (II-§1.28.17).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.statusID

public int statusID(int id, boolean load)

Calculates and returns the bitwise inclusive OR of the status of all the media tracked by this media tracker with the specified identifier. The possible flags are as above in the statusAll method (II-§1.28.16). An image that hasn't started loading has zero as its status.

If the load flag is true, then starts loading any images that are not yet being loaded.

Parameters:

id- the identifier of the images to check

load- if true, starts loading any images that are not yet being loaded

Returns:

the bitwise inclusive OR of the status of all of the media being tracked.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.waitForAll

public void waitForAll()
throws InterruptedException

Starts loading all images tracked by this media tracker. This method waits until all the images being tracked have finished loading.

If there is an error while loading or scaling an image then that image is considered finished loading. Use the isErrorAny method (II-§1.28.14) or isErrorID method (II-§1.28.15) to check for errors.

Throws

InterruptedException (I-§1.37)

If another thread has interrupted this thread.

See Also:

waitForID (II-§1.28.20).

public boolean waitForAll(long ms)
throws InterruptedException

Starts loading all images tracked by this media tracker. This method waits until all the images being tracked have finished loading, or until the length of time specified in milliseconds by the ms argument have passed.

If there is an error while loading or scaling an image then that image is considered finished loading. Use the isErrorAny method (II-§1.28.14) or isErrorID method (II-§1.28.15) to check for errors.

Parameters:

ms – the number of milliseconds to wait for the loading to complete

Returns:

true if all images were successfully loaded; false otherwise.

Throws

InterruptedException (I-§1.37)

If another thread has interrupted this thread.

See Also:

waitForID (II-§1.28.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MediaTracker.waitForID

public void waitForID(int id)
throws InterruptedException

Starts loading all images tracked by this media tracker with the specified identifier. This method waits until all the images with the specified identifier have finished loading.

If there is an error while loading or scaling an image then that image is considered finished loading. Use the isErrorAny method ([II-§1.28.14](#)) or isErrorID method ([II-§1.28.15](#)) to check for errors.

Parameters:

id– the identifier of the images to check

Throws

InterruptedException ([I-§1.37](#))

If another thread has interrupted this thread.

See Also:

waitForAll ([II-§1.28.18](#)).

public boolean waitForID(int id, long ms)
throws InterruptedException

Starts loading all images tracked by this media tracker with the specified identifier. This method waits until all the images with the specified identifier have finished loading, or until the length of time specified in milliseconds by the ms argument have passed.

If there is an error while loading or scaling an image then that image is considered finished loading. Use the isErrorAny method ([II-§1.28.14](#)) or isErrorID method ([II-§1.28.15](#)) to check for errors.

Parameters:

id– the identifier of the images to check

ms– the number of milliseconds to wait for the loading to complete

Returns:

true if all images were successfully loaded; false otherwise.

Throws

InterruptedException ([I-§1.37](#))

If another [thread](#) has interrupted this [thread](#).

See Also:

waitForAll ([II-§1.28.19](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.29 Class Menu

```
public class java.awt.Menu
    extends java.awt.MenuItem (II-§1.32)
    implements java.awt.MenuContainer (II-§1.44)
{
    // Constructors
    public Menu(String label); §1.29.1
    public Menu(String label, boolean tearOff); §1.29.2

    // Methods
    public MenuItem add(MenuItem mi); §1.29.3
    public void add(String label); §1.29.4
    public void addNotify(); §1.29.5
    public void addSeparator(); §1.29.6
    public int countItems(); §1.29.7
    public MenuItem getItem(int index); §1.29.8
    public boolean isTearOff(); §1.29.9
    public void remove(int index); §1.29.10
    public void remove(MenuComponent item); §1.29.11
    public void removeNotify(); §1.29.12
}
```

A menu is a pull-down component of a menu bar.

A menu can optionally be a *tear-off* menu. A tear-off menu can remain on the screen after the mouse button has been released. The mechanism for tearing off a menu is platform dependent.

Each item in a menu must belong to the **MenuItem** class (II-§1.32). This can be an instance of **MenuItem**, a submenu (an instance of **Menu**), or a check box (an instance of **CheckboxMenuItem** (II-§1.7)).

There is no event associated with clicking on a menu item to pull down the menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Menu.Menu

public Menu(String label)

Constructs a new menu with the specified label. This menu is not a tear-off menu.

Parameters:

label- the menu's label in the menu bar

public Menu(String label, boolean tearOff)

Constructs a new menu with the specified label.

If the tearOff argument is true, the menu can be torn off; it can remain on the screen after the mouse button has been released.

Parameters:

label- the menu's label in the menu bar

tearOff- if true, the menu is a tear-off menu

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.add

public MenuItem add(MenuItem mi)

Adds the specified menu item to this menu.

If the menu item had been part of another menu, removes it from that menu.

Parameters:

mi- the menu item to be added

Returns:

the menu item added.

public void add(String label)

Adds an item with the specified label to this menu. This method creates a menu item to hold the string.

Parameters:

label- the text on the item

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.addNotify

public void addNotify()

This method calls the createMenu method ([II-§1.41.14](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a MenuPeer ([II-§3.16](#)) for this menu. This peer allows the application to change the look of a menu without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class MenuItem ([II-§1.32.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.addSeparator

public void addSeparator()

Adds a separator line to this menu at the current position.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Menu.countItems

```
public int countItems()
```

Returns:

the number of elements in this menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Menu.getItem

public MenuItem getItem(int index)

Parameters:

index- the position of the item to be returned

Returns:

the item located at the specified index of this menu.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.isTearOff

public boolean isTearOff()

Returns:

true if this is a tear-off menu; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.remove

public void remove(int index)

Deletes the item at the specified index from this menu.

Parameters:

index- an index in the menu

public void remove(MenuComponent item)

Deletes the specified menu item from this menu. If the item is not part of the menu, nothing happens.

Parameters:

item- the menu component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Menu.removeNotify

public void removeNotify()

Notifies the menu to destroy its peer.

This menu also notifies each of its menu items to destroy their peers.

Overrides:

removeNotify in class MenuComponent (II-§1.31.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.30 Class MenuBar

```
public class java.awt.MenuBar
    extends java.awt.MenuComponent (II-§1.31)
    implements java.awt.MenuContainer (II-§1.44)
{
    // Constructors
    public MenuBar(); §1.30.1

    // Methods
    public Menu add(Menu m); §1.30.2
    public void addNotify(); §1.30.3
    public int countMenus(); §1.30.4
    public Menu getHelpMenu(); §1.30.5
    public Menu getMenu(int i); §1.30.6
    public void remove(int index); §1.30.7
    public void remove(MenuComponent m); §1.30.8
    public void removeNotify(); §1.30.9
    public void setHelpMenu(Menu m); §1.30.10
}
```

This class that encapsulates the platform's concept of a menu bar bound to a frame.

In order to attach the menu bar to a frame, the `setMenuBar` method (II-§1.19.28) in class `Frame` must be called.

This is what a menu bar looks like{ewc msdncd, EWGraphic, AWT0fa 0 /a "sunref.BMP"}

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.MenuBar

public MenuBar ()

Creates a new menu bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuBar.add

public Menu add(Menu m)

Adds the specified menu to this menu bar.

Parameters:

m – the menu to be added

Returns:

the menu added.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.addNotify

public void addNotify()

This method calls the createMenuBar method (II-§1.41.15) of this object's toolkit (II-§1.10.20) in order to create a MenuBarPeer (II-§3.13) for this menu bar. This peer allows the application to change the look of a menu bar without changing its functionality.

Most applications do not call this method directly.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuBar.countMenus

public int countMenus()

Returns:

the number of menus on this menu bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.getHelpMenu

public Menu getHelpMenu()

Returns:

the help menu on this menu bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar getMenu

public Menu getMenu(int i)

Parameters:

i – the position of the menu to be returned

Returns:

the menu at the specified index of this menu bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.remove

public void remove(int index)

Removes the menu located at the specified index from this menu bar.

Parameters:

index- the position of the menu to be removed

public void remove(MenuComponent m)

Removes the specified menu component from this menu bar.

Parameters:

m- the menu component to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.removeNotify

public void removeNotify()

Notifies this menu bar to destroy its peer.

This menu bar also notifies each of its menus to destroy their peers.

Overrides:

removeNotify in class MenuComponent (II-§1.31.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBar.setHelpMenu

public void setHelpMenu(Menu m)

Sets the help menu on this menu bar to be the specified menu.

Parameters:

m – the help menu

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.31 Class MenuComponent

```
public abstract class java.awt.MenuComponent
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public MenuComponent(); §1.31.1

    // Methods
    public Font getFont(); §1.31.2
    public MenuContainer getParent(); §1.31.3
    public MenuComponentPeer getPeer(); §1.31.4
    protected String  paramString(); §1.31.5
    public boolean postEvent(Event evt); §1.31.6
    public void removeNotify(); §1.31.7
    public void setFont(Font f); §1.31.8
    public String toString(); §1.31.9
}
```

The abstract class MenuComponent is the superclass of all menu related components.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuComponent.MenuComponent

public MenuComponent ()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.getFont

public `Font` `getFont()`

Returns:

the font used in this menu component, if there is one; null otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.getParent

public MenuContainer getParent()

Returns:

returns the menu component containing this menu component; null if this is the outermost component, the menu bar itself.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.getPeer

public MenuComponentPeer getPeer ()

Returns this menu component's peer. Every menu component has a peer associated with it. This peer allows the application to change the look of a menu component without changing its functionality.

Returns:

the menu component's peer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent paramString

protected String paramString()

Returns the parameter string representing the state of this menu component. This string is useful for debugging.

Returns:

the parameter string of this menu component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.postEvent

public **boolean** postEvent(**Event** evt)

Posts an event to this menu component by calling its handleEvent method. If handleEvent returns false, then posts the event to the menu component's parent.

Parameters:

evt- the event

Returns:

true if this menu component or one of its parents handled the event; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.removeNotify

public void removeNotify()

Notifies this menu component to destroy its peer.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.setFont

public void setFont(Font f)

Sets the font to be used for this menu component to the specified font. This font is also used by all subcomponents of the menu component, unless those subcomponents specify a different font.

Parameters:

f – the font to be set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuComponent.toString

Returns:

a string representation of this menu component.

Overrides:

toString in class Object ([I-§1.12.9](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.32 Class MenuItem

```
public class java.awt.MenuItem
    extends java.awt.MenuComponent (II-§1.31)
{
    // Constructors
    public MenuItem(String label); §1.32.1

    // Methods
    public void addNotify(); §1.32.2
    public void disable(); §1.32.3
    public void enable(); §1.32.4
    public void enable(boolean cond); §1.32.5
    public String getLabel(); §1.32.6
    public boolean isEnabled(); §1.32.7
    public String  paramString(); §1.32.8
    public void setLabel(String label); §1.32.9
}
```

All items in a menu must belong to MenuItem class or a subclass.

The default menu item represents a simple labeled menu item.

The picture of a menu bar (II-188) shows five menu items. The first two are simple menu items labelled "Basic" and "Simple." Following it is a separator (see §1.32.1). Next is a CheckboxMenuItem (II-§1.7) labelled "Check." Finally, there is a submenu labelled "More Examples," which is an instance of Menu (II-§1.29).

When a menu item is selected, the AWT sends an action event (II-§1.14.13) to the menu item's containing frame. The event's target is the menu item, and its object is the string label of the menu item. Note that the subclass Menu overrides this behavior and does not send any event to the frame until one of its subitems is selected.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuItem.MenuItem

public MenuItem(String label)

Constructs a new menu item with the specified label.

The label "-" is reserved to mean a separator between menu items. By default, all menu items except for separators are enabled.

Parameters:

label- the label for the menu item

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuItem.addNotify

public void addNotify()

This method calls the createMenuItem method ([II-§1.41.16](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a MenuItemPeer ([II-§3.15](#)) for this menu item. This peer allows the application to change the look of a menu item, without changing its functionality.

Most applications do not call this method directly.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuItem.disable

public void disable()

Disables this menu item. It can no longer be selected by the user.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuItem.enable

public void enable()

Enables this menu item; it can be selected by the user.

public void enable(boolean cond)

Enables this menu item if the flag is true; otherwise, disables it.

Parameters:

cond- if true, enable this menu item; otherwise disable it

See Also:

enable [\(II-§1.32.4\)](#)

disable [\(II-§1.32.3\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItem.getLabel

public String getLabel()

Returns:

the label of this menu item, or null if this menu item has no label.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItem.IsEnabled

public boolean isEnabled()

Checks whether this menu item is enabled.

Returns:

true if this menu item is enabled; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItem paramString

public String paramString()

Returns the parameter string representing the state of this menu item. This string is useful for debugging.

Returns:

the parameter string of this menu item.

Overrides:

paramString in class MenuComponent ([II-§1.31.5](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItem.setLabel

```
public void setLabel(String label)
```

Changes this menu item's label to be the label argument.

Parameters:

label- the new label, or null for no label

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.33 Class Panel

```
public class java.awt.Panel
    extends java.awt.Container (II-§1.11)
{
    // Constructors
    public Panel (); §1.33.1

    // Methods
    public void addNotify(); §1.33.2
}
```

A panel is the simplest container class. It provides space into which an application can attach any other component, including other panels.

The AWT sends the panel all mouse, keyboard, and focus events that occur over it.

The default layout manager for a panel is the FlowLayout (II-§1.16) layout manager.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Panel.Panel

public Panel ()

Creates a new panel. The default layout for a panel is FlowLayout (II-§1.16).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Panel.addNotify

public void addNotify()

This method calls the createPanel method ([II-§1.41.17](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a PanelPeer ([II-§3.17](#)) for this panel. This peer allows the application to change the look of a panel without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Container ([II-§1.11.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.34 Class Point

```
public class java.awt.Point
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public int x; §1.34.1
    public int y; §1.34.2

    // Constructors
    public Point(int x, int y); §1.34.3

    // Methods
    public boolean equals(Object obj); §1.34.4
    public int hashCode(); §1.34.5
    public void move(int x, int y); §1.34.6
    public String toString(); §1.34.7
    public void translate(int dx, int dy); §1.34.8
}
```

A point represents an (x, y) coordinate.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Point.x

public int x

The x coordinate.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.y

public int y

The y coordinate.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.Point

public Point(int x, int y)

Constructs and initializes a point to the specified (x, y) coordinate.

Parameters:

x- the x coordinate

y- the y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.equals

public **boolean** **equals**(**Object** obj)

The result is true if the argument is not null and is a Point object that has the same x and y coordinates as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Point.hashCode

public int hashCode()

Returns:

a hash code value for this point.

Overrides:

hashCode in class Object (I-§1.12.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.move

public void move(int x, int y)

Modifies this point so that it now represents the (x, y) coordinates indicated.

Parameters:

x- the new x coordinate

y- the new y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.toString

public String toString()

Returns:

a string representation of this point.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Point.translate

public void translate(int dx, int dy)

Translates this point by dx to the right and dy downward so that it now represents the point $(x + dx, y + dy)$, where it had been representing the point (x, y) .

Parameters:

dx- the amount to move this point to the right

dy- the amount to move this point downward

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.35 Class Polygon

```
public class java.awt.Polygon
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public int npoints; §1.35.1
    public int xpoints[]; §1.35.2
    public int ypoints[]; §1.35.3

    // Constructors
    public Polygon(); §1.35.4
    public Polygon(int xpoints[], int ypoints[], §1.35.5
        int npoints);

    // Methods
    public void addPoint(int x, int y); §1.35.6
    public Rectangle getBoundingBox(); §1.35.7
    public boolean inside(int x, int y); §1.35.8
}
```

A polygon consists of a list of (x, y) , where each successive pair of coordinates defines a side of the polygon.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Polygon.npoints

public int npoints

The total number of points.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.xpoints

public int xpoints[]

The array of x coordinates.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.ypoints

public int ypoints[]

The array of y coordinates.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.Polygon

public Polygon()

Creates an empty polygon.

public Polygon(int xpoints[], int ypoints[], int npoints)

Constructs and initializes a polygon from the specified parameters.

Parameters:

xpoints- an array of x coordinates

ypoints- an array of y coordinates

npoints- the total number of points in the polygon

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.addPoint

public void addPoint(int x, int y)

Appends a point to this polygon.

If an operation that calculates the bounding box of this polygon (getBoundingBox [\(II-§1.35.7\)](#) or inside [\(II-§1.35.8\)](#)) has already been performed, this method updates the bounding box.

Parameters:

x- the x coordinate of the point

y- the y coordinate of the point

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.getBoundingBox

public **Rectangle** **getBoundingBox()**

Returns:

the smallest rectangle than contains this polygon.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Polygon.inside

public boolean inside(int x, int y)

Determines if the specified point is inside this polygon. This method uses an even-odd insideness rule (also known as an "alternating rule") to determine whether the point (x, y) is inside this polygon.

It is based on code by Hanpeter van Vliet (hvvliet@inter.nl.net).

Parameters:

x- the xcoordinate of the point to be tested

y- the y coordinate of the point to be tested

Returns:

true if the point (x, y) is inside this polygon; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.36 Class Rectangle

```
public class java.awt.Rectangle
    extends java.lang.Object (l-§1.12)
{
    // Fields
    public int height; §1.36.1
    public int width; §1.36.2
    public int x; §1.36.3
    public int y; §1.36.4

    // Constructors
    public Rectangle(); §1.36.5
    public Rectangle(Dimension d); §1.36.6
    public Rectangle(int width, int height); §1.36.7
    public Rectangle(int x, int y, int width, int height); §1.36.8
    public Rectangle(Point p); §1.36.9
    public Rectangle(Point p, Dimension d); §1.36.10

    // Methods
    public void add(int newx, int newy); §1.36.11
    public void add(Point pt); §1.36.12
    public void add(Rectangle r); §1.36.13
    public boolean equals(Object obj); §1.36.14
    public void grow(int h, int v); §1.36.15
    public int hashCode(); §1.36.16
    public boolean inside(int x, int y); §1.36.17
    public Rectangle intersection(Rectangle r); §1.36.18
    public boolean intersects(Rectangle r); §1.36.19
    public boolean isEmpty(); §1.36.20
    public void move(int x, int y); §1.36.21
    public void reshape(int x, int y, int width, int height); §1.36.22
    public void resize(int width, int height); §1.36.23
    public String toString(); §1.36.24
    public void translate(int dx, int dy); §1.36.25
    public Rectangle union(Rectangle r); §1.36.26
}
```

A rectangle specifies an area defined by its top-left (x, y) coordinate, its width and its height.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.height

public int height

The height of the rectangle.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.width

public int width

The width of the rectangle.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.x

public int x

The x coordinate of the top-left corner of the rectangle.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.y

public int y

The y coordinate of the top-left corner of the rectangle.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.Rectangle

public Rectangle()

Constructs a new rectangle whose top-left corner is (0, 0), and whose width and height are 0.

public Rectangle(Dimension d)

Constructs a new rectangle whose top-left corner is (0, 0) and whose width and height are specified by the dimension argument.

Parameters:

d- the width and height

public Rectangle(int width, int height)

Constructs a new rectangle whose top-left corner is (0, 0) and whose width and height are the specified arguments.

Parameters:

width- the width of the rectangle

height- the height of the rectangle

public Rectangle(int x, int y, int width, int height)

Constructs a new rectangle whose top-left corner is (x, y) and whose width and height are the specified arguments.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the rectangle

height- the height of the rectangle

public **Rectangle**(**Point** p)

Constructs a new rectangle whose top-left corner is the specified point argument and whose width and height are 0.

Parameters:

p- the top-left corner of the rectangle

public **Rectangle**(**Point** p, **Dimension** d)

Constructs a new rectangle whose top-left corner is the specified point argument and whose width and height are specified by the dimension argument.

Parameters:

p- the top-left corner of the rectangle

d- the width and height

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.add

public void add(int newx, int newy)

Adds the point (newx, newy) to this rectangle.

This rectangle is modified to be the smallest rectangle that contains both this rectangle and the point.

Parameters:

newx- the x coordinate of the new point

newy- the y coordinate of the new point

public void add(Point pt)

Adds the point to this rectangle.

This rectangle is modified to be the smallest rectangle that contains both this rectangle and the point.

Parameters:

pt- a point

public void add(Rectangle r)

Adds the rectangle argument to this rectangle.

This rectangle is modified to be the smallest rectangle that contains both rectangles.

Parameters:

r- a rectangle

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.equals

public **boolean** **equals**(**Object** obj)

The result is true if the argument is not null and is a Rectangle object that has the same top-left corner, width, and height as this object.

Parameters:

obj – the object to compare with

Returns:

true if the objects are the same; false otherwise.

Overrides:

equals in class Object ([I-§1.12.3](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Rectangle.grow

public void grow(int h, int v)

Modifies the rectangle so that it is h units larger on both the left and right side, and v units larger at both the top and bottom.

The new rectangle has $(x - h, y - v)$ as its top-left corner, a width of $x + 2h$ and a height of $y + 2v$.

Parameters:

h- the horizontal expansion

v- the vertical expansion

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.hashCode

public int hashCode()

Returns:

a hash code value for this object.

Overrides:

hashCode in class Object ([I-§1.12.6](#)).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Rectangle.inside

public boolean inside(int x, int y)

Checks if the specified point lies inside this rectangle.

Parameters:

x- the x coordinate

y- the y coordinate

Returns:

true if the point (x, y) is inside this rectangle; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.intersection

public Rectangle intersection(Rectangle r)

Parameters:

r- a rectangle

Returns:

the largest rectangle contained in both the rectangle argument and in this rectangle.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Rectangle.intersects

public boolean intersects(Rectangle r)

Determines if this rectangle and the rectangle argument intersect. Two rectangles intersect if their intersection is non-empty.

Parameters:

r- a rectangle

Returns:

true if the rectangle argument and this rectangle intersect; false otherwise.

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Rectangle.isEmpty

public boolean isEmpty()

Determines if this rectangle is empty. A rectangle is empty if its width or its height is less than or equal to zero.

Returns:

true if this rectangle is empty; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.move

public void move(int x, int y)

Moves this rectangle so that its new top-left corner is the specified (x, y) coordinate.

Parameters:

x- the new x coordinate

y- the new y coordinate

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.reshape

public void reshape(int x, int y, int width, int height)

Reshapes this rectangle so that its new top-left corner is the specified (x, y) coordinate and its new width and height are the specified arguments.

Parameters:

x- the new x coordinate

y- the new y coordinate

width- the new width

height- the new height

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.resize

public void resize(int width, int height)

Resizes this rectangle so that its new width and height are the indicated arguments.

Parameters:

width- the new width

height- the new height

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.toString

public String toString()

Returns:

a string representation of this rectangle.

Overrides:

toString in class Object ([I-§1.12.9](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.translate

public void translate(int dx, int dy)

Translates this rectangle by dx to the right and dy downward so that its top-left corner is now point $(x + dx, y + dy)$, where it had been the point (x, y) .

Parameters:

dx- the amount to move the rectangle to the right

dy- the amount to move the rectangle downward

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Rectangle.union

public Rectangle union(Rectangle r)

Computes the union of this rectangle with the argument rectangle.

Parameters:

r- a rectangle

Returns:

the smallest rectangle containing both the rectangle argument and this rectangle.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.37 Class Scrollbar

```
public class java.awt.Scrollbar
    extends java.awt.Component (II-§1.10)
{
    // Fields
    public final static int HORIZONTAL; §1.37.1
    public final static int VERTICAL; §1.37.2

    // Constructors
    public Scrollbar(); §1.37.3
    public Scrollbar(int orientation); §1.37.4
    public Scrollbar(int orientation, int value, §1.37.5
        int visible, int minimum, int maximum);

    // Methods
    public void addNotify(); §1.37.6
    public int getLineIncrement(); §1.37.7
    public int getMaximum(); §1.37.8
    public int getMinimum(); §1.37.9
    public int getOrientation(); §1.37.10
    public int getPageIncrement(); §1.37.11
    public int getValue(); §1.37.12
    public int getVisible(); §1.37.13
    protected String paramString(); §1.37.14
    public void setLineIncrement(int l); §1.37.15
    public void setPageIncrement(int l); §1.37.16
    public void setValue(int value); §1.37.17
    public void setValues(int value, int visible, §1.37.18
        int minimum, int maximum);
}
```

A scroll bar provides a convenient means of allowing a user to select from a range of values. For example, the following three scrollbars could be used to pick the each of the red, green, and blue components of a color{ewc msdncd, EWGraphic, AWT0fh 0 /a "sunref.BMP"}:

Each scroll bar was created with the code like the following:

```
redSlider=new Scrollbar(Scrollbar.VERTICAL, 0, 1, 0, 255);
add(redSlider);
```

Alternatively, a scroll bar can represent a range of values. For example, if using a scroll bar for scrolling through text, the width of the "bubble" can represent the amount of text visible. Here is an example of a scrollbar representing a range:{ewc msdncd, EWGraphic, AWT0fh 1 /a "sunref.BMP"}

The value range represented by the bubble is the *visible* amount of the scroll bar.

The code to produce the above scroll bar is


```
ranger = new Scrollbar(Scrollbar.HORIZONTAL, 0, 64, 0, 255);
add(ranger);
```

Note that the maximum value above, 255, is the maximum value for the "left side" of the scroll bar.

In addition, whenever the user changes the value of the scroll bar, AWT sends one of the following five events to the user:

- a scroll absolute event (§5.14.28) if the user drags the bubble
- a scroll line down (§5.14.29) event if the user clicks in the right arrow of a horizontal scroll bar or the bottom arrow of a vertical scroll bar
- a scroll line up event (II-§1.14.30) if the user clicks in the left arrow of a horizontal scroll bar or the top arrow of a vertical scroll bar
- a scroll page down (§5.14.31) event if the user clicks to the right of the bubble in a horizontal scroll bar or below the bubble in a vertical scroll bar
- a scroll page up (II-§1.14.32) event if the user clicks to the left of the bubble in a horizontal scroll bar or above the bubble in a vertical scroll bar

The event's target is the scroll bar, and its object is an Integer (I-§1.8) giving the value represented by the scroll bar.

If an application wants to perform some action when the value in a scroll bar is changed, it must override handleEvent method of the scroll bar or of one of its containing windows. The code to perform that should be of the following form:

```
public boolean handleEvent(Event event) {
    if (event.target == scrollbar) {
        do something
        return true;
    } else {
        return super.handleEvent(event);
    }
}
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Scrollbar.HORIZONTAL

```
public final static int HORIZONTAL = 0
```

Constant indicating to construct a horizontal scroll bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Scrollbar.VERTICAL

```
public final static int VERTICAL = 1
```

Constant indicating to construct a vertical scroll bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Scrollbar.Scrollbar

public Scrollbar()

Constructs a new vertical scroll bar.

public Scrollbar(int orientation)

Constructs a new scroll bar with the specified orientation.

The orientation argument must be one of the two values, Scrollbar.HORIZONTAL or (II-§1.37.1) Scrollbar.VERTICAL (II-§1.37.2), indicating a horizontal or vertical scroll bar, respectively.

Parameters:

orientation- indicates the orientation of the scroll bar

Throws

IllegalArgumentException (I-§1.32)

When an illegal orientation is given.

public Scrollbar(int orientation, int value, int visible, int minimum, int maximum)

Constructs a new scroll bar with the specified orientation, initial value, page size, and minimum and maximum values.

The orientation argument must be one of the two values Scrollbar.HORIZONTAL or (II-§1.37.1) Scrollbar.VERTICAL (II-§1.37.2), indicating a horizontal or vertical scroll bar, respectively.

If the specified maximum value is less than the minimum value, it is changed to be the same as the minimum value. If the initial value is lower than the minimum value, it is changed to be the minimum value; if it is greater than the maximum value, it is changed to be the maximum value.

Parameters:

orientation- indicates the orientation of the scroll bar

value- the initial value of the scroll bar

visible- the size represented by the bubble in the scroll bar; the scroll bar uses this value when paging up or down by a page.

minimum- the minimum value of the scroll bar

maximum- the maximum value of the scroll bar

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Scrollbar.addNotify

public void addNotify()

This method calls the createScrollbar method (II-§1.41.18) of this object's toolkit (II-§1.10.20) in order to create a ScrollbarPeer (II-§3.18) for this button. This peer allows the application to change the look of a scroll bar without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.getLineIncrement

public int getLineIncrement()

Determines the line increment of this scroll bar, which is the amount that is added or subtracted from this scroll bar's value when the user clicks the down or up gadget.

Returns:

the line increment of this scroll bar.

See Also:

setLineIncrement [\(II-§1.37.15\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.Maximum

public int `getMaximum()`

Returns:

the maximum value of this scroll bar.

See Also:

`getMinimum` ([II-§1.37.9](#))

`getValue` ([II-§1.37.12](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.getMinimum

public int getMinimum()

Returns:

the maximum value of this scroll bar.

See Also:

getMaximum ([II-§1.37.8](#))

getValue ([II-§1.37.12](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.getOrientation

public int getOrientation()

Determines the orientation of this scroll bar. The value returned is either HORIZONTAL (II-§1.37.1) or VERTICAL (II-§1.37.2).

Returns:

the orientation of this scroll bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.getPageIncrement

public int getPageIncrement()

Determines the page increment of this scroll bar, which is the amount that is added to or subtracted from this scroll bar' value when the user clicks the page down or page up gadget.

Returns:

the page increment for this scroll bar.

See Also:

getPageIncrement [\(II-§1.37.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.GetValue

public int GetValue()

Returns:

the current value of this scroll bar.

See Also:

getMinimum [\(II-§1.37.9\)](#)
getMaximum [\(II-§1.37.8\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.getVisible

public int getVisible()

Determines the "visible" amount of this scroll bar, which is the range of values represented by the width of the bubble in this scroll bar.

Returns:

the "visible" amount of this scroll bar.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.paramString

protected String paramString()

Returns the parameter string representing the state of this scroll bar. This string is useful for debugging.

Returns:

the parameter string of this scroll bar.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.setLineIncrement

public void setLineIncrement(int l)

Sets the line increment of this scroll bar.

The line increment is the value that is added to or subtracted from the value of this scroll bar when the user clicks the line down or line up gadget.

Parameters:

l – the new line increment

See Also:

getLineIncrement [\(II-§1.37.7\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.setPageIncrement

public void setPageIncrement(int l)

Sets the page increment of this scroll bar.

The page increment is the value that is added to or subtracted from the value of the scroll bar when the user clicks the page down or page up gadget.

Parameters:

l – the new page increment

See Also:

getPageIncrement ([II-§1.37.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.setValue

public void setValue(int value)

Sets the value of this scroll bar to the specified value. If the specified value is below this scroll bar's current minimum or above the current maximum, it becomes the minimum or maximum value, respectively.

Parameters:

value- the new value of this scroll bar

See Also:

getValue [\(II-§1.37.12\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Scrollbar.setValues

```
public void setValues(int value, int visible,  
int minimum, int maximum)
```

Sets several parameters of this scroll bar simultaneously.

Parameters:

value- the value of this scroll bar

visible- the amount visible per page

minimum- the minimum value of this scroll bar

maximum- the maximum value of this scroll bar

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.38 Class TextArea

```
public class java.awt.TextArea
    extends java.awt.TextComponent (II-§1.39)
{
    // Constructors
    public TextArea(); §1.38.1
    public TextArea(int rows, int cols); §1.38.2
    public TextArea(String text); §1.38.3
    public TextArea(String text, int rows, int cols); §1.38.4

    // Methods
    public void addNotify(); §1.38.5
    public void appendText(String str); §1.38.6
    public int getColumns(); §1.38.7
    public int getRows(); §1.38.8
    public void insertText(String str, int pos); §1.38.9
    public Dimension minimumSize(); §1.38.10
    public Dimension minimumSize(int rows, int cols); §1.38.11
    protected String paramString(); §1.38.12
    public Dimension preferredSize(); §1.38.13
    public Dimension preferredSize(int rows, int cols); §1.38.14
    public void replaceText(String str, int start, int end); §1.38.15
}
```

A text area is a multiline area for displaying text. It can be set to allow editing or to be read-only.

For example, the following code:

```
new TextArea("Hello", 5, 40);
```

produces the text area shown below:

```
{ewc msdn cd, EWGraphic, AWT0fi 0 /a "sunref.BMP"}.
```

When ¹the text area receives or loses the focus, AWT sends a "got focus" (II-§1.14.12) or "lost focus" (II-§1.14.20) event to the text area. An application should override the gotFocus method (II-§1.10.21) or the lostFocus method (II-§1.10.39) of the text area in order to cause some action to occur.

Most user interfaces use an external event (such as clicking on a button) to trigger an action on the text area.

```
{ewl msdn cd.dll, ewcright, /c"Microsoft"}
```


TextArea.TextArea

public TextArea()

Constructs a new text area.

public TextArea(int rows, int cols)

Constructs a new text area with the specified number of rows and columns.

Parameters:

rows- the number of rows

cols- the number of columns

public TextArea(String text)

Constructs a new text area with the specified text displayed.

Parameters:

text- the text to be displayed

public TextArea(String text, int rows, int cols)

Constructs a new TextArea with the specified text, and the specified number of rows and columns.

Parameters:

text- the text to be displayed

rows- the number of rows

cols- the number of columns

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.addNotify

public void addNotify()

This method calls the createTextArea method (II-§1.41.19) of this object's toolkit (II-§1.10.20) in order to create a TextAreaPeer (II-§3.19) for this text area. This peer allows the application to change the look of a text area without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component (II-§1.10.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.appendText

public void appendText(String str)

Appends the given text to this text area's current text.

Parameters:

str- the text to append

See Also:

insertText [\(II-§1.38.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.setColumns

```
public int getColumns()
```

Returns:

the number of columns in this text area.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextArea.getRows

public int getRows ()

Returns:

the number of rows in this text area.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.insertText

public void insertText(String str, int pos)

Inserts the specified text at the specified position in this text area.

Parameters:

str- the text to insert

pos- the position at which to insert the text

See Also:

setText in class TextComponent ([II-§1.39.11](#))

replaceText ([II-§1.38.15](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.minimumSize

public **Dimension** **minimumSize()**

Determines the minimum size of this text area. If the application has specified both the number of rows and the number of columns for this text area, and both are greater than zero, then the text area's peer's `minimumSize` method ([II-§3.19.2](#)) is called with the number of rows and columns in order to determine the minimum size.

If the text area does not have a peer, or if the number of rows or number of columns specified by the application is less than or equal to zero, the superclass's `minimumSize` [method \(II-§1.10.40\)](#) is called to determine the minimum size.

Returns:

the minimum dimensions needed for this text area.

Overrides:

`minimumSize` in class `Component` ([II-§1.10.40](#)).

public **Dimension** **minimumSize(int rows, int cols)**

Determines the minimum size of a text area with the specified number of rows and columns. This text area's peer's `minimumSize` [method \(II-§3.19.2\)](#) is called with the number of rows and columns in order to determine the minimum size.

If this text area does not have a peer, the superclass's `minimumSize` [method \(II-§1.10.40\)](#) is called to determine the minimum size.

Parameters:

`rows`- the number of rows

`cols`- the number of columns

Returns:

the minimum dimensions needed to display the text area with the specified number of rows and columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea paramString

protected String paramString()

Returns the parameter string representing the state of this text area. This string is useful for debugging.

Returns:

the parameter string of this text area.

Overrides:

paramString in class TextComponent (II-§1.39.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.preferredSize

public **Dimension** preferredSize()

Determines the preferred size of this text area. If the application has specified both the number of rows and the number of columns for this text area, and both are greater than zero, then the text area's peer's preferredSize method ([II-§3.19.3](#)) is called with the number of rows and columns in order to determine the preferred size.

If this text area does not have a peer, or if the number of rows or number of columns specified by the application is less than or equal to zero, the superclass's preferredSize [method \(II-§1.10.53\)](#) is called to determine the preferred size.

Returns:

the preferred dimensions needed for this text area.

Overrides:

preferredSize in class Component ([II-§1.10.53](#)).

public **Dimension** preferredSize(int rows, int cols)

Determines the preferred size of a text area with the specified number of rows and columns. This text area's peer's preferredSize [method \(II-§3.19.3\)](#) is called with the number of rows and columns in order to determine the preferred size.

If this text area does not have a peer the superclass's preferredSize [method \(II-§1.10.53\)](#) is called to determine the preferred size.

Parameters:

`rows`- the number of rows

`cols`- the number of columns

Returns:

the preferred dimensions needed to display the text area with the specified number of rows and columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextArea.replaceText

public void replaceText(String str, int start, int end)

Replaces the text in the text area from the start (inclusive) index to the end (exclusive) index with the new text specified.

Parameters:

str- the replacement text

start- the start position

end- the end position

See Also:

insertText [\(II-§1.38.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse, keyboard, or focus events to a text area. In Java 1.1, the AWT sends the text area all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.39 Class TextComponent

```
public class java.awt.TextComponent
    extends java.awt.Component (II-§1.10)
{
    // Methods
    public String getSelectedText(); §1.39.1
    public int getSelectionEnd(); §1.39.2
    public int getSelectionStart(); §1.39.3
    public String getText(); §1.39.4
    public boolean isEditable(); §1.39.5
    protected String  paramString(); §1.39.6
    public void removeNotify(); §1.39.7
    public void select(int selStart, int selEnd); §1.39.8
    public void selectAll(); §1.39.9
    public void setEditable(boolean t); §1.39.10
    public void setText(String t); §1.39.11
}
```

A text component is the superclass of any component that allows the editing of some text.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextComponent.getSelectedText

public String `getSelectedText()`

Returns:

the selected text in this text component.

See Also:

`setText` ([II-§1.39.11](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.getSelectionEnd

public int `getSelectionEnd()`

Returns:

selected text's end position in this text component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.getSelectionStart

```
public int getSelectionStart()
```

Returns:

the selected text's start position in this text component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextComponent.getText

public String **getText**()

Returns:

the text of this text component.

See Also:

setText [\(II-§1.39.11\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.setEditable

public boolean isEditable()

Returns:

true if this text component is editable; false otherwise.

See Also:

setEditable ([II-§1.39.10](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent paramString

protected String paramString()

Returns the parameter string representing the state of this text component. This string is useful for debugging.

Returns:

the parameter string of this text component.

Overrides:

paramString in class Component ([II-§1.10.51](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.removeNotify

public void removeNotify()

Notifies this text component to destroy its peer.

Overrides:

removeNotify in class Component (II-§1.10.58).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.select

public void select(int selStart, int selEnd)

Selects the text in this text component found from the specified start (inclusive) index to the specified end index (exclusive).

Parameters:

selStart- the start position of the text to select

selEnd- the end position of the text to select

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.selectAll

```
public void selectAll()
```

Selects all the text in this text component.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextComponent.setEditable

public void **setEditable**(boolean t)

If the boolean argument is true, this text component becomes user editable. If the flag is false, the user cannot change the text of this text component.

Parameters:

t- a flag indicating whether the text component should become user editable

See Also:

isEditable (II-§1.39.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponent.setText

public void setText(String t)

Sets the text of this text component to be the specified text.

Parameters:

t– the new text

See Also:

getText [\(II-§1.39.4\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.40 Class TextField

```
public class java.awt.TextField
    extends java.awt.TextComponent (II-§1.39)
{
    // Constructors
    public TextField(); §1.40.1
    public TextField(int cols); §1.40.2
    public TextField(String text); §1.40.3
    public TextField(String text, int cols); §1.40.4

    // Methods
    public void addNotify(); §1.40.5
    public boolean echoCharIsSet(); §1.40.6
    public int getColumns(); §1.40.7
    public char getEchoChar(); §1.40.8
    public Dimension minimumSize(); §1.40.9
    public Dimension minimumSize(int cols); §1.40.10
    protected String paramString(); §1.40.11
    public Dimension preferredSize(); §1.40.12
    public Dimension preferredSize(int cols); §1.40.13
    public void setEchoCharacter(char c); §1.40.14
}
```

A text field is a component that presents the user with a single editable line of text.

For example, the following code:

```
TextField tf1, tf2, tf3, tf4;
// a blank text field
tf1 = new TextField();
// blank field of 20 columns
tf2 = new TextField(20);
// Predefined text displayed
tf3 = new TextField("Hello!");
// Predefined text in 30 columns
tf4 = new TextField("Hello", 30);
```

produces the text fields shown below:

```
{ewc msdncd, EWGraphic, AWT0fk 0 /a "sunref.BMP"}
```

Every time the user types a key¹ in the text field, AWT sends a key press event (II-§1.14.15) and a key release event (II-§1.14.16) to the text field. These event's target field is the button, and their key field is the key typed.

In addition, whether the user types the return key, AWT sends an action event (II-§1.14.11) to the text field. This event's target is the text field, and its object is the string contents of the text field;

the string will not have a return character as its last character. An application should override the action method (II-§1.10.1) of the text field or of one of its containing windows in order to cause some action to occur.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextField.TextField

public TextField()

Constructs a new text field.

public TextField(int cols)

Constructs a new text field the specified number of characters wide.

Parameters:

cols- the number of characters

public TextField(String text)

Constructs a new text field initialized with the specified text.

Parameters:

text- the text to be displayed

public TextField(String text, int cols)

Constructs a new text field initialized with the specified text and wide enough to hold the specified number of characters.

Parameters:

text- the text to be displayed

cols- the number of characters

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.addNotify

public void addNotify()

This method calls the createTextField method ([II-§1.41.20](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a TextFieldPeer ([II-§3.21](#)) for this text field. This peer allows the application to change the look of a text field without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Component ([II-§1.10.2](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.echoCharIsSet

public boolean echoCharIsSet()

Returns:

true if this text field has a character set for echoing; false otherwise.

See Also:

setEchoCharacter ([II-§1.40.14](#))

getEchoChar ([II-§1.40.8](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.setColumns

```
public int getColumns()
```

Returns:

the number of columns in this text field.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextField.getEchoChar

public char getEchoChar()

Returns:

the echo character for this text field.

See Also:

setEchoCharacter ([II-§1.40.14](#))

echoCharIsSet ([II-§1.40.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.minimumSize

public **Dimension** **minimumSize()**

Determines the minimum size of this text field. If the application has specified the number of columns for this text field, and it is greater than zero, then this text field's peer's **minimumSize** method (II-§3.21.1) is called with the number columns in order to determine the minimum size.

If this text field does not have a peer, or if the number of columns specified by the application is less than or equal to zero, the superclass's **minimumSize** method (II-§1.10.40) is called to determine the minimum size.

Returns:

the minimum dimensions needed for this text area.

Overrides:

minimumSize in class **Component** (II-§1.10.40).

public **Dimension** **minimumSize(int cols)**

Determines the minimum size of a text field with the specified number of columns. This text field's peer's **minimumSize** method (II-§3.21.1) is called with the number of columns in order to determine the minimum size.

If this text field does not have a peer the superclass's **minimumSize** method (II-§1.10.40) is called to determine the minimum size.

Parameters:

cols- the number of columns

Returns:

the minimum dimensions needed to display a text area with the specified number of columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField paramString

protected String paramString()

Returns the parameter string representing the state of this text field. This string is useful for debugging.

Returns:

the parameter string of this text field.

Overrides:

paramString in class TextComponent (II-§1.39.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.preferredSize

public Dimension preferredSize()

Determines the preferred size of this text field. If the application has specified the number of columns for this text field, and if it is greater than zero, then this text field's peer's preferredSize method (II-§3.21.2) is called with the number of columns in order to determine the preferred size.

If this text field does not have a peer, or if the number of columns specified by the application is less than or equal to zero, the superclass's preferredSize method (II-§1.10.53) is called to determine the preferred size.

Returns:

the preferred dimensions needed for this text field.

Overrides:

preferredSize in class Component (II-§1.10.53).

public Dimension preferredSize(int cols)

Determines the preferred size of a text field with the specified number of columns. This text field's peer's preferredSize method (II-§3.21.2) is called with the number columns in order to determine the preferred size.

If this text field does not have a peer the superclass's preferredSize method (II-§1.10.53) is called to determine the preferred size.

Parameters:

cols- the number of columns

Returns:

the preferred dimensions needed to display the text field with the specified number of columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextField.setEchoCharacter

public void setEchoCharacter(char c)

Sets the echo character for this text field. Any character that the user types in the text field is echoed in this text field as the echo character.

An echo character is useful for fields where the user input shouldn't be echoed to the screen such as in the case of a text field for typing in a password.

Parameters:

c – the echo character for this text field

See Also:

echoCharIsSet [\(II-§1.40.6\)](#)

getEchoChar [\(II-§1.40.8\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Footnotes

¹In Java 1.0, the AWT does not send mouse or focus events to a text area. In Java 1.1, the AWT sends the text area all mouse, keyboard, and focus events that occur over it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.41 Class Toolkit

```
public abstract class java.awt.Toolkit
    extends java.lang.Object (l-§1.12)
{
    // Constructors
    public Toolkit(); §1.41.1

    // Methods
    public abstract int §1.41.2
        checkImage(Image image, int width, int height,
                    ImageObserver observer);

    protected abstract ButtonPeer §1.41.3
        createButton(Button target);
    protected abstract CanvasPeer §1.41.4
        createCanvas(Canvas target);
    protected abstract CheckboxPeer §1.41.5
        createCheckbox(Checkbox target);
    protected abstract CheckboxMenuItemPeer §1.41.6
        createCheckboxMenuItem(CheckboxMenuItem target);
    protected abstract ChoicePeer §1.41.7
        createChoice(Choice target);
    protected abstract DialogPeer §1.41.8
        createDialog(Dialog target);
    protected abstract FileDialogPeer §1.41.9
        createFileDialog(FileDialog target);
    protected abstract FramePeer createFrame(Frame target); §1.41.10
    public abstract Image §1.41.11
        createImage(ImageProducer producer);
    protected abstract LabelPeer createLabel(Label target); §1.41.12
    protected abstract ListPeer createList(List target); §1.41.13
    protected abstract MenuPeer createMenu(Menu target); §1.41.14
    protected abstract MenuBarPeer §1.41.15
        createMenuBar(MenuBar target);
    protected abstract MenuItemPeer §1.41.16
        createMenuItem(MenuItem target);
    protected abstract PanelPeer createPanel(Panel target); §1.41.17
    protected abstract ScrollbarPeer §1.41.18
        createScrollbar(Scrollbar target);
    protected abstract TextAreaPeer §1.41.19
        createTextArea(TextArea target);
    protected abstract TextFieldPeer §1.41.20
        createTextField(TextField target);
    protected abstract WindowPeer §1.41.21
        createWindow(Window target);
    public abstract ColorModel getColorModel(); §1.41.22
```



```

    public static Toolkit getDefaultToolkit(); §1.41.23
    public abstract String[] getFontList(); §1.41.24
    public abstract FontMetrics §1.41.25
        getFontMetrics(Font font);
    public abstract Image getImage(String filename); §1.41.26
    public abstract Image getImage(URL url); §1.41.27
    public abstract int getScreenResolution(); §1.41.28
    public abstract Dimension getScreenSize(); §1.41.29
    public abstract boolean §1.41.30
        prepareImage(Image image, int width, int height,
            ImageObserver observer);
    public abstract void sync(); §1.41.31
}

```

This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of the class are used to bind the various components to particular native toolkit implementations.

Most applications should not call any of the methods in this class directly. These methods are called by the addNotify methods of the various components in the Abstract Window Toolkit.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Toolkit.Toolkit

public Toolkit()

The default constructor for a toolkit.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.checkImage

public abstract int

`checkImage(Image image, int width, int height, ImageObserver observer)`

If the width and height arguments are both -1, this method returns the status of the construction of a screen representation of the specified image in this toolkit. Otherwise, this method returns the status of the construction of a scaled representation of the specified image at the specified width and height.

This method does not cause the image to begin loading. An application must use the `prepareImage` ([II-§1.41.30](#)) method to force the loading of an image.

This toolkit method is used by the `checkImage` methods ([II-§1.10.4](#), [§1.10.5](#)) of `Component`.

Information on the flags returned by this method can be found in [II-§2.11](#).

Parameters:

`image`- the image whose status is being checked

`width`- the width of the scaled version to check the status of

`height`- the height of the scaled version to check the status of

`observer`- the `ImageObserver` object to be notified as the image is being prepared

Returns:

the bitwise inclusive OR of the `ImageObserver` ([II-§2.11](#)) flags indicating what information about the image is available.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createButton

protected abstract ButtonPeer createButton(Button target)

Parameters:

`target` - the button to be implemented

Returns:

this toolkit's implementation of a Button (II-§1.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createCanvas

protected abstract CanvasPeer createCanvas(Canvas target)

Parameters:

target– the canvas to be implemented

Returns:

this toolkit's implementation of a Canvas (II-§1.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createCheckbox

protected abstract CheckboxPeer

createCheckbox (Checkbox target)

Parameters:

target - the check box to be implemented

Returns:

this toolkit's implementation of a Checkbox (II-§1.5).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createCheckboxMenuItem

protected abstract CheckboxMenuItemPeer

createCheckboxMenuItem (CheckboxMenuItem target)

Parameters:

target - the check box menu item to be implemented

Returns:

this toolkit's implementation of a CheckboxMenuItem (II-§1.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createChoice

protected abstract ChoicePeer createChoice (Choice target)

Parameters:

`target` - the choice list to be implemented

Returns:

this toolkit's implementation of a Choice (II-§1.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createDialog

protected abstract DialogPeer createDialog(Dialog target)

Parameters:

`target`– the dialog window to be implemented

Returns:

this toolkit's implementation of a Dialog (II-§1.12).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createFileDialog

protected abstract FileDialogPeer

createFileDialog(FileDialog target)

Parameters:

target - the file dialog window to be implemented

Returns:

this toolkit's implementation of a FileDialog (II-§1.15).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createFrame

protected abstract FramePeer createFrame (Frame target)

Parameters:

`target` - the frame to be implemented

Returns:

this toolkit's implementation of a Frame (II-§1.19).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createImage

public abstract Image createImage(ImageProducer producer)

Parameters:

target- the image to be implemented

Returns:

this toolkit's implementation of an Image (II-§1.24).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createLabel

protected abstract LabelPeer createLabel (Label target)

Parameters:

target- the label to be implemented

Returns:

this toolkit's implementation of a Label (II-§1.26).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createList

protected abstract ListPeer createList(List target)

Parameters:

target- the scrolling list to be implemented

Returns:

this toolkit's implementation of a List (II-§1.27).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createMenu

protected abstract MenuPeer createMenu (Menu target)

Parameters:

target- the menu to be implemented

Returns:

this toolkit's implementation of a Menu (II-§1.29).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createMenuBar

protected abstract MenuBarPeer

createMenuBar (MenuBar target)

Parameters:

target- the menu bar to be implemented

Returns:

this toolkit's implementation of a MenuBar (II-§1.30).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createMenuItem

protected abstract MenuItemPeer

createMenuItem(MenuItem target)

Parameters:

target - the menu item to be implemented

Returns:

this toolkit's implementation of a MenuItem (II-§1.32).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createPanel

protected abstract PanelPeer createPanel (Panel target)

Parameters:

`target` - the panel to be implemented

Returns:

this toolkit's of a Panel (II-§1.33).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

Toolkit.createScrollbar

protected abstract ScrollbarPeer

createScrollbar(Scrollbar target)

Parameters:

target - the scroll bar to be implemented

Returns:

this toolkit's implementation of a ScrollBar (II-§1.37).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createTextArea

protected abstract TextAreaPeer

createTextArea (TextArea target)

Parameters:

target - the text area to be implemented

Returns:

this toolkit's implementation of a TextArea (II-§1.38).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createTextField

protected abstract TextFieldPeer

createTextField(TextField target)

Parameters:

target - the text field to be implemented

Returns:

this toolkit's implementation of a TextField (II-§1.40).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.createWindow

protected abstract WindowPeer createWindow(Window target)

Parameters:

`target` - the window to be implemented

Returns:

this toolkit's implementation of a Window (II-§1.42).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getColorModel

public abstract ColorModel getColorModel()

Determine's the color model of this toolkit's screen.

The ColorModel ([II-§2.1](#)) is an abstract class that encapsulates how to translate between pixel values of an image and its red, green, blue, and alpha components.

This toolkit method is used by the getColorModel method ([II-§1.10.13](#)) of Component.

Returns:

the color model of the toolkit's screen.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getDefaultToolkit

public static Toolkit getDefaultToolkit()

Gets the default toolkit.

If there is a system property named "awt.toolkit", that property is treated as the name of a class that is a subclass of Toolkit.

If the system property does not exist, then the default toolkit used is the class named "sun.awt.motif.MToolkit", which is a motif implementation of the Abstract Window Toolkit.

Returns:

the default toolkit.

Throws

AWTError (II-§1.46)

If a toolkit could not be found or could not be instantiated.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getFontList

```
public abstract String[] getFontList()
```

Returns:

the names of the available fonts in this toolkit.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Toolkit.getFontMetrics

public abstract FontMetrics getFontMetrics(Font font)

Parameters:

font- a font

Returns:

the screen metrics of the font argument in this toolkit.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getImage

public abstract Image getImage(String filename)

Parameters:

filename- a file containing pixel data in a recognized file format

Returns:

an image which gets its pixel data from the specified file.

public abstract Image getImage(URL url)

Parameters:

url- a URL which specifies an image.

Returns:

an image which gets its pixel data from the specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getScreenResolution

public abstract int getScreenResolution()

Returns:

this toolkit's screen resolution in dots-per-inch.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.getScreenSize

public abstract Dimension getScreenSize()

Returns:

the size of this toolkit's screen in pixels.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.prepareImage

public abstract boolean

prepareImage(Image image, int width, int height, ImageObserver observer)

Prepares an image for rendering. If the width and height arguments are both -1, this method prepares the image for rendering on the default screen; otherwise, this method prepares an image for rendering on the default screen at the specified width and height.

The image data is downloaded asynchronously in another thread and an appropriately scaled screen representation of the image is generated.

This toolkit method is used by the prepareImage methods (II-§1.10.54, §1.10.55) of Component.

Information on the flags returned by this method can be found in II-§2.11.

Parameters:

image- the image to prepare a screen representation for
width- the width of the desired screen representation
height- the height of the desired screen representation
observer- the ImageObserver object to be notified as the image is being prepared

Returns:

true if the image has already been fully prepared; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Toolkit.sync

public abstract void sync()

Synchronizes this toolkit's graphics state. Some window systems may do buffering of graphics events. This method ensures that the display is up to date.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.42 Class Window

```
public class java.awt.Window
    extends java.awt.Container (II-§1.11)
{
    // Constructors
    public Window(Frame parent); §1.42.1

    // Methods
    public void addNotify(); §1.42.2
    public void dispose(); §1.42.3
    public Toolkit getToolkit(); §1.42.4
    public final String getWarningString(); §1.42.5
    public void pack(); §1.42.6
    public void show(); §1.42.7
    public void toBack(); §1.42.8
    public void toFront(); §1.42.9
}
```

A Window is a top-level window; it has no borders and no menu bar. It could be used, for example, to implement a pop-up menu. The AWT sends the window all mouse, keyboard, and focus events that occur over it.

The default layout for a window is BorderLayout (II-§1.1).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Window.Window

public Window(Frame parent)

Constructs a new invisible window.

The window behaves as a modal dialog in that it will block input to other application windows when shown.

Use the show method ([II-§1.42.7](#)) to cause the window to become visible.

Parameters:

parent- the main application frame

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Window.addNotify

public void addNotify()

This method calls the createWindow method ([II-§1.41.21](#)) of this object's toolkit ([II-§1.10.20](#)) in order to create a WindowPeer ([II-§3.22](#)) for this window. This peer allows the application to change the look of a window without changing its functionality.

Most applications do not call this method directly.

Overrides:

addNotify in class Container ([II-§1.11.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.dispose

public void dispose()

Disposes of this window and any resources used by this window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.getToolkit

public Toolkit getToolkit()

Determines the toolkit of this window.

The implementation of getToolkit in class Window returns the default toolkit (II-§1.41.23). However subclasses of Window can override this method in order to create their own toolkits.

Returns:

the toolkit of this window.

Overrides:

getToolkit in class Component (II-§1.10.20).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.getWarningString

public final String getWarningString()

Returns the warning string that is displayed with this window. If this window is insecure, the warning string is displayed somewhere in the visible area of the window. A window is insecure if there is a security manager, and its checkTopLevelWindow method (I-§1.15.22) returns false when passed this window as an argument.

If the window is secure, then the getWarningString method returns null. If the window is insecure, this methods checks for a system property awt.appletWarning and returns the string value of that property. If there is no such property, the default warning string is used instead. The default warning string is "Warning: Applet Window."

Returns:

the warning string for this window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.pack

public void pack()

Causes the subcomponents of this window to be laid out at their preferred size.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Window.show

public void show()

If this window is not yet visible, make it visible. If this window is already visible, then bring it to the front (II-§1.42.9).

Overrides:

show in class Component (II-§1.10.70).

See Also:

hide in class Component (II-§1.10.23).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.toBack

public void toBack()

Sends this window to the back.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Window.toFront

public void toFront()

Brings this window to the front.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.43 Interface LayoutManager

```
public interface java.awt.LayoutManager
{
    // Methods
    public abstract void addLayoutComponent(String name, §1.43.1
        Component comp);
    public abstract void layoutContainer(Container parent); §1.43.2
    public abstract Dimension §1.43.3
        minimumLayoutSize(Container parent);
    public abstract Dimension §1.43.4
        preferredLayoutSize(Container parent);
    public abstract void §1.43.5
        removeLayoutComponent(Component comp);
}
```

The LayoutManager interface specifies the methods that all layout managers must implement.

A layout manager is a class for laying out the components of a Container (II-§1.11)

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


LayoutManager.addLayoutComponent

public abstract void

addLayoutComponent(String name, Component comp)

Adds the specified component to this layout using the indicated tag.

Most applications do not call this method directly. This method is called when a component is added to a container using the add(String, Component) method (II-§1.11.3).

Parameters:

name- a tag understood by this layout manager

comp- the component to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LayoutManager.layoutContainer

public abstract void layoutContainer(Container parent)

Lays out the container argument using this layout

This method may reshape the components in the specified target container in order to satisfy the constraints of this layout manager.

Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

parent- the container in which to do the layout.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LayoutManager.minimumLayoutSize

public abstract Dimension

minimumLayoutSize(Container parent)

Determines the minimum size of the container argument using this layout manager. Most applications do not call this method directly. This method is called when a container calls its layout method (II-§1.11.11).

Parameters:

parent- the container in which to do the layout

Returns:

the minimum dimensions needed to lay out the subcomponents of the specified container.

See Also:

preferredLayoutSize (II-§1.43.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LayoutManager.preferredLayoutSize

public abstract Dimension

preferredLayoutSize(Container parent)

Determines the preferred size of the container argument using this layout manager. Most applications do not call this method directly. This method is called when a container calls its preferredSize method (II-§1.11.17).

Parameters:

parent- the container in which to do the layout

Returns:

the preferred dimensions to lay out the subcomponents of the specified container.

See Also:

minimumLayoutSize (II-§1.43.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LayoutManager.removeLayoutComponent

public abstract void

removeLayoutComponent(Component comp)

Removes the specified component from this layout.

Most applications do not call this method directly. This method is called when a container calls its remove ([II-§1.11.19](#)) or removeAll ([II-§1.11.20](#)) methods.

Parameters:

comp- the component to be removed

{ewl msdncl.dll, ewcright, /c"Microsoft"}

§1.44 Interface MenuContainer

```
public interface java.awt.MenuContainer
{
    // Methods
    public abstract Font getFont(); §1.44.1
    public abstract boolean postEvent(Event evt); §1.44.2
    public abstract void remove(MenuComponent comp); §1.44.3
}
```

The MenuContainer interface specifies the methods that all menu-related containers must implement. Note that menu containers are not required to be full-fledged Container objects (II-§1.11).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuContainer.getFont

public **abstract** **Font** **getFont()**

Returns:

the font used in this menu component, if there is one; null otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuContainer.postEvent

public abstract boolean postEvent(Event evt)

Posts an event to this menu container.

Parameters:

evt- the event

Returns:

true if this menu component or one of its parents handled the event; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuContainer.remove

public abstract void remove(MenuComponent comp)

Removes the specified menu component from this menu container.

Parameters:

m- the menu component to be removed

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§1.45 Class AWTException

```
public class java.awt.AWTException
    extends java.lang.Exception (I-§1.30)
{
    // Constructors
    public AWTException(String msg); §1.45.1
}
```

Thrown when an AWTException has occurred.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AWTException.AWTException

public AWTException(String msg)

Constructs an AWTException with the specified detail message.

Parameters:

msg- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§1.46 Class AWTError

```
public class java.awt.AWTError
    extends java.lang.Error (I-§1.48)
{
    // Constructors
    public AWTError(String msg); §1.46.1
}
```

Thrown when a serious AWTError has occurred.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AWTError.AWTError

public AWTError(String msg)

Constructs an AWTError with the specified detail message.

Parameters:

msg- the detail message

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.awt.image

Classes

- 2.1 Class ColorModel
- 2.2 Class CropImageFilter
- 2.3 Class DirectColorModel
- 2.4 Class FilteredImageSource
- 2.5 Class ImageFilter
- 2.6 Class IndexColorModel
- 2.7 Class MemoryImageSource
- 2.8 Class PixelGrabber
- 2.9 Class RGBImageFilter

Interfaces

- 2.10 Interface ImageConsumer
- 2.11 Interface ImageObserver
- 2.12 Interface ImageProducer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.1 Class ColorModel

```
public abstract class java.awt.image.ColorModel
    extends java.lang.Object (I-§1.12)
{
    // Fields
    protected int pixel_bits; §2.1.1

    // Constructors
    public ColorModel(int bits); §2.1.2

    // Methods
    public abstract int getAlpha(int pixel); §2.1.3
    public abstract int getBlue(int pixel); §2.1.4
    public abstract int getGreen(int pixel); §2.1.5
    public int getPixelSize(); §2.1.6
    public abstract int getRed(int pixel); §2.1.7
    public int getRGB(int pixel); §2.1.8
    public static ColorModel getRGBdefault(); §2.1.9
}
```

This abstract class is the superclass for all classes that encapsulate methods for translating from pixel values to their alpha (transparency), red, green, and blue components.

The java.awt.image classes **IndexColorModel** (II-§2.6) and **DirectColorModel** (II-§2.3) are subclasses of this class.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ColorModel.pixel_bits

protected int pixel_bits

The number of bits per pixel.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.ColorModel

public ColorModel(int bits)

Constructs a ColorModel which describes a pixel with the specified number of bits.

Parameters:

`bits`— the number of bits per pixel

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getAlpha

public abstract int getAlpha(int pixel)

Determines the alpha transparency of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates that the pixel is completely transparent. The value 255 indicates that the pixel is opaque.

Parameters:

pixel- a pixel value

Returns:

the alpha transparency represented by the pixel value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getBlue

public abstract int getBlue(int pixel)

Determines the blue component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the blue color component represented by the pixel value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getGreen

public abstract int getGreen(int pixel)

Determines the green component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

The green color component ranging from 0 to 255.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getPixelSize

```
public int getPixelSize()
```

Returns:

the number of bits per pixel in this color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ColorModel.getRed

public abstract int getRed(int pixel)

Determines the red component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the red color component ranging from 0 to 255.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getRGB

public int getRGB(int pixel)

Calculates a single integer representing the alpha, red, green, and blue components of a pixel in this color model. The components are each scaled to be a value between 0 and 255 . The integer returned is the number such that bits 24-31 are the alpha value, 16-23 are the red value, bits 8-15 are the green value, and bits 0-7 are the blue value.

Parameters:

pixel- a pixel value

Returns:

an integer representing this color in RGB format.

See Also:

getRGBdefault (II-§2.1.9).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ColorModel.getRGBdefault

public static ColorModel getRGBdefault()

Returns the default Abstract Window Toolkit color model.

The Abstract Window Toolkit represents each pixel as a 32-bit integer. Bits 24-31 are the alpha transparency, bits 16-23 are the red value, bits 8-15 are the green value, and bits 0-7 are the blue value.

This method returns a ColorModel object which describes that pixel format and can be used to extract alpha, red, green, and blue values from such color values.

Returns:

the default Abstract Window Toolkit color model.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.2 Class CropImageFilter

```
public class java.awt.image.CropImageFilter
    extends java.awt.image.ImageFilter (II-§2.5)
{
    // Constructors
    public CropImageFilter(int x, int y, int w, int h); §2.2.1

    // Methods
    public void setDimensions(int w, int h); §2.2.2
    public void setPixels(int x, int y, int w, int h, §2.2.3
        ColorModel model, byte pixels[],
        int off, int scansize);
    public void setPixels(int x, int y, int w, int h, §2.2.4
        ColorModel model, int pixels[],
        int off, int scansize);
    public void setProperties(Hashtable props); §2.2.5
}
```

The cropped image filter is an image filter for cropping images. This class extends the basic ImageFilter class (II-§2.5) to extract a given rectangular region of an existing image and provides a source for new image containing only the extracted region.

This class is meant to be used in conjunction with a FilteredImageSource (II-§2.4) to produce cropped versions of existing images.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CropImageFilter.CropImageFilter

public CropImageFilter(int x, int y, int w, int h)

Constructs a cropped image filter that extracts the absolute rectangular region of pixels from its source image as specified by the x, y, w, and h parameters.

Parameters:

- x- the x location of the top of the rectangle to be extracted
- y- the y location of the top of the rectangle to be extracted
- w- the width of the rectangle to be extracted
- h- the height of the rectangle to be extracted

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CropImageFilter.setDimensions

public void setDimensions(int w, int h)

The image producer calls the setDimensions of the image consumer to tell it the width and height of the image.

The setDimensions method of CroppedImageFilter ignores its arguments. It calls the setDimensions method (II-§2.10.12) of its image consumer with the width and height arguments of the constructor (II-§2.2.1).

Parameters:

width- the width of the image

height- the height of the image

Overrides:

setDimensions in class ImageFilter (II-§2.5.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CropImageFilter.setPixels

public void

```
setPixels(int x, int y, int w, int h, ColorModel model, byte
pixels[], int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.14.

The setPixels method of CroppedImageFilter determines if the specified rectangle intersects its cropping region. If so, it calls the setPixels method (II-§2.10.14) after modifying the x, y, w, h, and offset arguments to reflect only the intersecting region.

Parameters:

x- left coordinate of rectangle
y- top coordinte of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

Overrides:

setPixels in class ImageFilter (II-§2.5.10).

public void

```
setPixels(int x, int y, int w, int h, ColorModel model, int pixels[],
int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.15.

The setPixels method of CroppedImageFilter determines if the specified rectangle intersects

its cropping region. If so, it calls the setPixels method (II-§2.10.15) after modifying the x, y, w, h, and offset arguments to reflect only the intersecting region.

Parameters:

x- left coordinate of rectangle
y- top coordinte of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

Overrides:

setPixels in class ImageFilter (II-§2.5.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CropImageFilter.setProperties

public void setProperties(Hashtable props)

The image producer calls the setProperties method of the image consumer to let it know of additional properties of the image. For more information on this method and its arguments, see §2.10.16.

The setProperties method of CroppedImageFilter adds the property "cropect" with the value new Rectangle(x, y, width, height) to the table of properties, and then calls the setProperties method of its image consumer (II-§2.10.16) with the modified properties table.

Parameters:

props- a hash table that maps image properties to their value

Overrides:

setProperties in class ImageFilter (II-§2.5.12).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.3 Class DirectColorModel

```
public class java.awt.image.DirectColorModel
    extends java.awt.image.ColorModel (II-§2.1)
{
    // Constructors
    public DirectColorModel(int bits, int rmask, int gmask, §2.3.1
                           int bmask);
    public DirectColorModel(int bits, int rmask, int gmask, §2.3.2
                           int bmask, int amask);

    // Methods
    public final int getAlpha(int pixel); §2.3.3
    public final int getAlphaMask(); §2.3.4
    public final int getBlue(int pixel); §2.3.5
    public final int getBlueMask(); §2.3.6
    public final int getGreen(int pixel); §2.3.7
    public final int getGreenMask(); §2.3.8
    public final int getRed(int pixel); §2.3.9
    public final int getRedMask(); §2.3.10
    public final int getRGB(int pixel); §2.3.11
}
```

The direct color model is a color model (II-§2.1) which specifies a translation from pixel values to alpha, red, green, and blue components using the actual bits of the pixel value. This color model is similar to an X11 TrueColor visual.

Many of the methods in this class are final: the underlying native graphics code makes assumptions about the layout and operation of this class and those assumptions are reflected in the implementations of the methods here that are marked final. Applications can subclass this class for other reasons, but they cannot override or modify the behavior of the final methods.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.DirectColorModel

public DirectColorModel(int bits, int rmask, int gmask, int bmask)

Constructs a direct color model in which each of the given masks specify which bits in the pixels contain the red, green, and blue components.

Pixels described by this color model all have alpha components of 255, indicating that they are fully opaque.

Each of the bit masks must be contiguous, and must be smaller than 2^{bits} .

Parameters:

bits- the number of bits in a pixel

rmask- the bits in the pixel representing the red component

gmask- the bits in the pixel representing the green component

bmask- the bits in the pixel representing the blue component

public

DirectColorModel(int bits, int rmask, int gmask,
int bmask, int amask)

Constructs a direct color model in which each of the given masks specify which bits in the pixels contain the alpha, red, green, and blue components.

Each of the bit masks must be contiguous, and must be smaller than 2^{bits} .

Parameters:

bits- the number of bits in a pixel

rmask- the bits in the pixel representing the red component

gmask- the bits in the pixel representing the green component

bmask- the bits in the pixel representing the blue component

`amask`– the bits in the pixel representing the alpha component

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.getAlpha

public final int getAlpha(int pixel)

Determines the alpha transparency of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates that the pixel is completely transparent. The value 255 indicates that the pixel is opaque.

Parameters:

pixel- a pixel value

Returns:

the alpha transparency represented by the pixel value.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DirectColorModel.getAlphaMask

```
public final int getAlphaMask()
```

Returns:

a mask indicating which bits in a pixel contain the alpha transparency component in this color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.getBlue

public final int getBlue(int pixel)

Determines the blue component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the blue color component represented by the pixel value.

Overrides:

getBlue in class ColorModel ([II-§2.1.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DirectColorModel.getBlueMask

```
public final int getBlueMask()
```

Returns:

a mask indicating which bits in a pixel contain the blue color component in this color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.getGreen

public final int getGreen(int pixel)

Determines the green component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the blue color component represented by the pixel value.

Overrides:

getGreen in class ColorModel [\(II-§2.1.5\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DirectColorModel.getGreenMask

```
public final int getGreenMask()
```

Returns:

a mask indicating which bits in a pixel contain the green color component in this color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.getRed

public final int getRed(int pixel)

Determines the red component of a pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the red color component represented by the pixel value.

Overrides:

getRed in class ColorModel ([II-§2.1.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DirectColorModel.getRedMask

```
public final int getRedMask()
```

Returns:

a mask indicating which bits in a pixel contain the red color component in this color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DirectColorModel.getRGB

public **final** **int** **getRGB**(**int** **pixel**)

Calculates a single integer representing the alpha, red, green, and blue components of the pixel in this color model. The components are each scaled to be a value between 0 and 255. The integer returned is the number such that bits 24-31 are the alpha value, 16-23 are the red value, bits 8-15 are the green value, and bits 0-7 are the blue value.

This format is the pixel format of the default RGB colormodel (II-§2.1.9).

Parameters:

`pixel` - a pixel value

Returns:

an integer representing this color in RGB format.

Overrides:

`getRGB` in class `ColorModel` (II-§2.1.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.4 Class FilteredImageSource

```
public class java.awt.image.FilteredImageSource
    extends java.lang.Object (I-§1.12)
    implements java.awt.image.ImageProducer (II-§2.12)
{
    // Constructors
    public FilteredImageSource(ImageProducer orig, §2.4.1
        ImageFilter imgf);

    // Methods
    public void addConsumer(ImageConsumer ic); §2.4.2
    public boolean isConsumer(ImageConsumer ic); §2.4.3
    public void removeConsumer(ImageConsumer ic); §2.4.4
    public void §2.4.5
        requestTopDownLeftRightResend(ImageConsumer ic);
    public void startProduction(ImageConsumer ic); §2.4.6
}
```

A filtered image source is an implementation of the image producer interface (II-§2.12) which takes an existing image and an image filter (II-§2.5) and uses them to produce a new filtered version of the original image.

Here is an example which filters an image by swapping the red and blue components:

```
Image src = getImage("doc:///demo/images/duke/T1.gif");
// see comments in §2.9 for an implementation of RedBlueSwapFilter

ImageFilter colorfilter = new RedBlueSwapFilter();
Image img = createImage(new FilteredImageSource(src.getSource(),
    colorfilter));
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FilteredImageSource.FilteredImageSource

public FilteredImageSource(ImageProducer orig, ImageFilter imgf)

Constructs a new image producer from an existing image producer and an an image filter (II-§2.5)

Parameters:

orig- an existing image producer

imgf- an image filter

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilteredImageSource.addConsumer

public void addConsumer(ImageConsumer ic)

Registers the image consumer ([II-§2.10](#)) argument as wanting the image produced by this image producer. For more information on this method and its arguments, see the addConsumer method([II-§2.12.1](#)) of ImageProducer.

The addConsumer method of FilteredImageSource calls its image filter's getFilterInstance method ([II-§2.5.4](#)) to create a new filter instance for the image consumer argument. The resulting filter instance is then passed as an argument to the image producer's addConsumer method ([II-§2.12.1](#)).

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilteredImageSource.isConsumer

public boolean isConsumer(ImageConsumer ic)

Parameters:

ic- an image consumer

Returns:

true if the specified image consumer argument (II-§2.10) is currently registered with this image producer as one of its consumers; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilteredImageSource.removeConsumer

public void removeConsumer(ImageConsumer ic)

Removes the specified image consumer ([II-§2.10](#)) **object** from the list of consumers registered to receive the image data from this image producer. For more information on this **method** and its arguments, see [§2.12.3](#).

The removeConsumer **method** of FilteredImageSource calls its image producer's removeConsumer **method** ([II-§2.12.3](#)) with the image filter that was created when this image consumer was first registered ([§2.4.2](#), [§2.4.6](#)).

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilteredImageSource.requestTopDownLeftRightResend

public void

requestTopDownLeftRightResend (ImageConsumer ic)

An image consumer sends this message to request that the image producer attempt to resend the image data one more time in top-down, left-to-right order. For more information on this method and its arguments, see §2.12.4.

The requestTopDownLeftRightResend method of FilteredImageSource calls its image producer's requestTopDownLeftRightResend method (II-§2.12.4) with the image filter that was created when this image consumer was first registered (§2.4.2, §2.4.6).

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FilteredImageSource.startProduction

public void startProduction(ImageConsumer ic)

Registers the image consumer ([II-§2.10](#)) **argument** as wanting the image produced by this image producer. In addition, this **method** forces the image producer to start an immediate reconstruction of the image data. For more information on this **method** and its arguments, see the startProduction **method** ([II-§2.12.5](#)) of ImageProducer.

The startProduction **method** of FilteredImageSource calls its image filter's getFilterInstance **method** ([II-§2.5.4](#)) to create a new filter instance for the image consumer **argument**. The resulting filter instance is then passed as an **argument** to the image producer's startProduction **method** ([II-§2.12.5](#)).

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.5 Class ImageFilter

```
public class java.awt.image.ImageFilter
    extends java.lang.Object (I-§1.12)
    implements java.awt.image.ImageConsumer (II-§2.10),
               java.lang.Cloneable (I-§1.22)
{
    // Fields
    protected ImageConsumer consumer; §2.5.1

    // Constructors
    public ImageFilter(); §2.5.2

    // Methods
    public Object clone(); §2.5.3
    public ImageFilter getFilterInstance(ImageConsumer ic) §2.5.4
    public void imageComplete(int status); §2.5.5
    public void resendTopDownLeftRight(ImageProducer ip); §2.5.6
    public void setColorModel(ColorModel model); §2.5.7
    public void setDimensions(int width, int height); §2.5.8
    public void setHints(int hints); §2.5.9
    public void setPixels(int x, int y, int w, int h, §2.5.10
        ColorModel model, byte pixels[],
        int off, int scansize);
    public void setPixels(int x, int y, int w, int h, §2.5.11
        ColorModel model, int pixels[],
        int off, int scansize);
    public void setProperties(Hashtable props); §2.5.12
}
```

This class is the superclass of all classes that are meant to filter the data delivered from an ImageProducer (II-§2.12) to an ImageConsumer (II-§2.10).

This class and its subclasses are meant to be used in conjunction with a FilteredImageSource (II-§2.4) object to produce filtered versions of existing images.

The image filter implemented by this class is the "null filter" which has no effect on the data passing through. Filters should subclass this class and override the methods in order to modify the data as necessary.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageFilter.consumer

protected ImageConsumer consumer

The image consumer (II-§2.10) of the particular image data stream for which this image filter is filtering data.

The field is not initialized by the constructor, but by the call to the getFilterInstance method (II-§2.5.4), when the FilteredImageSource (II-§2.4) is creating a unique instance of this object for a particular image data stream.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.ImageFilter

public ImageFilter()

The default constructor for this method.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.clone

public Object clone()

Returns:

a clone of this object.

Overrides:

clone in class Object (I-§1.12.2).

{ewl msdncl.dll, ewcright, /c"Microsoft"}

ImageFilter.getFilterInstance

public ImageFilter getFilterInstance(ImageConsumer ic)

Creates an image filter which filters the image for the specified image consumer.

The getFilterInstance method of ImageFilter clones the object and sets its consumer field to the image consumer argument. Subclasses can override this method, if they require more setup than this.

Parameters:

ic- the image consumer

Returns:

a unique instance of an image filter object which actually performs the filtering for the specified image consumer (II-§2.10).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.imageComplete

public void imageComplete(int status)

The image producer calls the imageComplete method when it has completed an image or it has encountered an error in producing or loading the image. For more information on this method and its status argument, see §2.10.10.

The imageComplete method of ImageFilter calls the imageComplete method (II-§2.10.10) of the image consumer with the identical argument.

Parameters:

status- the status of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.resendTopDownLeftRight

public void resendTopDownLeftRight(ImageProducer ip)

An image consumer calls the resendTopDownLeftRight of its producer to request that the pixel data be resent in that order. For more information on this method see §2.12.4.

An image filter can respond to this request in one of three ways:

- The filter can forward this request to the image producer (II-§2.12.4) using itself as the requesting image consumer. This is the default behavior provided by the resendTopDownLeftRight method of ImageFilter.
- The filter can resend the pixels in the right order on its own (presumably because the generated pixels have been saved in some sort of buffer).
- The filter can ignore this request.

Parameters:

ip- the image producer that is feeding this instance of the filter

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.setColorModel

public void setColorModel (ColorModel model)

The image producer calls the setColorModel method to specify the color model for the majority of the subsequent setPixels method calls. For more information on this method and its model argument, see §2.10.11.

The setColorModel method of ImageFilter calls the setColorModel method (II-§2.10.11) of the image consumer with the identical argument.

Parameters:

model- a color map used in subsequent setPixel calls

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.setDimensions

public void setDimensions(int width, int height)

The image producer calls the setDimensions of the image consumer to tell it the width and height of the image.

The setDimensions method of ImageFilter calls the setDimensions method (II-§2.10.12) of its image consumer with the identical arguments.

Parameters:

width- the width of the image

height- the height of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.setHints

public void setHints(int hints)

The image producer calls the setHints method of the image consumer to indicate the order in which the bits are to be delivered. For more information on the hints passed to the image consumer, see §2.10.13.

The setHints method of ImageFilter calls the setHints method (II-§2.10.13) of its image consumer with the identical argument.

Parameters:

hints- hints about the order in which the bits are to be delivered

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.setPixels

public void

```
setPixels(int x, int y, int w, int h, ColorModel model,  
          byte pixels[], int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.14.

The setPixels method of ImageFilter calls the setPixels method (II-§2.10.14) of its image consumer with the identical arguments.

Parameters:

x- left coordinate of rectangle
y- top coordinate of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

public void

```
setPixels(int x, int y, int w, int h, ColorModel model, int  
pixels[], int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.15.

The setPixels method of ImageFilter calls the setPixels method (II-§2.10.15) of its image consumer with the identical arguments.

Parameters:

x- left coordinate of rectangle
y- top coordinate of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageFilter.setProperties

public void setProperties(Hashtable props)

The image producer calls the setProperties method of the image consumer to let it know of additional properties of the image. For more information on this method and its arguments, see §2.10.16.

The setProperties method of ImageFilter calls the setProperties method of its image consumer §2.10.16 with the properties argument after modifying it slightly:

- If the hash table already has no key "filters," a string representation of the current filter is added to the hash table under the key "filter."
- If such a key is already in the hash table, a string representation of the current filter is appended to the property (which must be a String) already stored under that key, and stores this value back in the hash table.

Parameters:

props- a hash table that maps image properties to their value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.6 Class IndexColorModel

```
public class java.awt.image.IndexColorModel
    extends java.awt.image.ColorModel (II-§2.1)
{
    // Constructors
    public IndexColorModel(int bits, int size, §2.6.1
        byte r[], byte g[], byte b[]);
    public IndexColorModel(int bits, int size, §2.6.2
        byte r[], byte g[],
        byte b[], byte a[]);
    public IndexColorModel(int bits, int size, §2.6.3
        byte r[], byte g[],
        byte b[], int trans);
    public IndexColorModel(int bits, int size, §2.6.4
        byte cmap[], int start,
        boolean hasalpha);
    public IndexColorModel(int bits, int size, §2.6.5
        byte cmap[], int start,
        boolean hasalpha, int trans);

    // Methods
    public final int getAlpha(int pixel); §2.6.6
    public final void getAlphas(byte a[]); §2.6.7
    public final int getBlue(int pixel); §2.6.8
    public final void getBlues(byte b[]); §2.6.9
    public final int getGreen(int pixel); §2.6.10
    public final void getGreens(byte g[]); §2.6.11
    public final int getMapSize(); §2.6.12
    public final int getRed(int pixel); §2.6.13
    public final void getReds(byte r[]); §2.6.14
    public final int getRGB(int pixel); §2.6.15
    public final int getTransparentPixel(); §2.6.16
}
```

The index color model class specifies a color model (II-§2.1) in which a pixel value is converted into alpha, red, green, and blue components by using the pixel value as an index into a color map. Each entry in the color map gives the alpha, red, green, and blue components for the corresponding pixel.

An optional *transparent* pixel can be specified. This pixel is completely transparent, independent of the alpha value recorded for that pixel value.

The maximum size of the color map is 256 entries.

This color model is similar to an X11 PseudoColor visual.

Many of the methods in this class are final; the underlying native graphics code makes assumptions about the layout and operation of this class and those assumptions are reflected in the implementations of the methods here that are marked final. Applications can subclass this class for other reasons, but they cannot override or modify the behavior of the final methods.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IndexColorModel.IndexColorModel

```
public IndexColorModel(int bits, int size, byte r[], byte g[], byte b[])
```

Constructs an IndexColorModel from the given arrays of red, green, and blue components.

Pixels described by this color model all have alpha components of 255 (fully opaque).

Each of the three arrays must have at least size elements, and it must be the case that $size < 2^{bits}$. These first size elements of the arrays are the red, green, and blue values for pixels in the range $0 \leq i < size$. Pixels in the range $size \leq i < 2^{bits}$ have red, green, and blue values of 0.

Parameters:

- bits- the number of bits in a pixel
- size- the size of the color component arrays
- r- an array of red color components
- g- an array of green color components
- b- an array of blue color components

```
public IndexColorModel(int bits, int size, byte r[], byte g[], byte b[], byte a[])
```

Constructs an IndexColorModel from the given arrays of red, green, blue and alpha components.

Each of the four arrays must have at least size elements, and it must be the case that $size < 2^{bits}$. These first size elements of the arrays are the red, green, and blue values for pixels in the range $0 \leq i < size$. Pixels in the range $size \leq i < 2^{bits}$ have red, green, and blue, and alpha values of 0.

Parameters:

- bits- the number of bits in a pixel

size- a lower bound on the size of each of the arrays

r- an array of red color components

g- an array of green color components

b- an array of blue color components

a- an array of alpha value components

```
public IndexColorModel(int bits, int size, byte r[], byte g[], byte  
b[], int trans)
```

Constructs an IndexColorModel from the given arrays of red, green, and blue components.

Pixels described by this color model all have alpha components of 255 (fully opaque), except for the transparent pixel.

Each of the three arrays must have at least size elements, and it must be the case that $size < 2^{bits}$. These first size elements of the arrays are the red, green, and blue values for pixels in the range $0 \leq i < size$. Pixels in the range $size \leq i < 2^{bits}$ have red, green, and blue values of 0.

Parameters:

bits- the number of bits in a pixel

size- the size of the color component arrays

r- an array of red color components

g- an array of green color components

b- an array of blue color components

trans- the index of the transparent pixel

public

```
IndexColorModel(int bits, int size, byte cmap[], int start, boolean  
hasalpha)
```

Constructs an index color model from a single array of packed red, green, blue and optional alpha components.

If the `hasalpha` argument is false, then the `cmap` array must have length of at least `start + 3 * size`. The red, green, and blue components for a pixel in the range `0 <= i < size` are in the three elements of the `cmap` array starting at `cmap[start + 3i]`. Its alpha component is 255 (fully opaque).

If the `hasalpha` argument is true, then the `cmap` array must have length of at least `start + 4 * size`. The red, green, blue, and alpha components for a pixel in the range `0 <= i < size` are in the four elements of the `cmap` array starting at `cmap[start + 4i]`.

Pixels in the range `size <= i < 2bits` have red, green, and blue values of 0. Their alpha component is 0 if `hasalpha` is true, and 255 otherwise.

Parameters:

`bits`- the number of bits in a pixel
`size`- the size of the color component arrays
`cmap`- an array of color components
`start`- the starting offset of the first color component
`hasalpha`- if true, the alpha values are contained in the `cmap` array

```
public IndexColorModel(int bits, int size, byte cmap[], int start,  
boolean hasalpha, int trans)
```

Constructs an index color model from a single array of packed red, green, blue, and optional alpha values.

The color model is constructed the same way as described in [§2.6.4](#). In addition, the specified transparent index represents a pixel which is considered entirely transparent regardless of any alpha value specified for it. The array must have enough values in it to fill all of the needed component arrays of the specified size.

Parameters:

`bits`- the number of bits in a pixel
`size`- the size of the color component arrays
`cmap`- an array of color components
`start`- the starting offset of the first color component

`hasalpha`- if true, the alpha values are in the cmap array

`trans`- the index of the fully transparent pixel

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

IndexColorModel.getAlpha

public final int getAlpha(int pixel)

Determines the alpha transparency value of the pixel in this color model. The value ranges from 0 to 255. The value 0 indicates that the pixel is completely transparent. The value 255 indicates that the pixel is opaque.

Parameters:

pixel- a pixel value

Returns:

The alpha transparency represented by the pixel value.

Overrides:

getAlpha in class ColorModel ([II-§2.1.3](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getAlphas

`public final void getAlphas(byte a[])`

Copies the array of alpha transparency values from this color model into the given array.

Only the initial entries of the array as specified by the getMapSize method (II-§2.6.12) are written.

Parameters:

a- an array into which to place the results

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getBlue

public final int getBlue(int pixel)

Determines the blue component of the pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the blue color component represented by the pixel value.

Overrides:

getBlue in class ColorModel ([II-§2.1.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getBlues

`public final void getBlues(byte b[])`

Copies the array of blue color components from this color model into the given array.

Only the initial entries of the array as specified by the getMapSize method (II-§2.6.12) are written.

Parameters:

b- an array into which to place the results

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getGreen

public **final** **int** **getGreen**(**int** **pixel**)

Determines the green component of the pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel— a pixel value

Returns:

the blue color component represented by the pixel value.

Overrides:

getGreen in class ColorModel [\(II-§2.1.5\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getGreens

`public final void getGreens(byte g[])`

Copies the array of green color components from this color model into the given array.

Only the initial entries of the array as specified by the getMapSize method (II-§2.6.12) are written.

Parameters:

g- an array into which to place the results

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getMapSize

```
public final int getMapSize()
```

Returns:

the value of the size argument when creating this index color model.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


IndexColorModel.getRed

public final int getRed(int pixel)

Determines the red component of the pixel in this color model. The value ranges from 0 to 255. The value 0 indicates no contribution from this primary color. The value 255 indicates the maximum intensity of this color component.

Parameters:

pixel- a pixel value

Returns:

the red color component represented by the pixel value.

Overrides:

getRed in class ColorModel ([II-§2.1.7](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getReds

`public final void getReds(byte r[])`

Copies the array of red color components from this color model into the given array.

Only the initial entries of the array as specified by the getMapSize method (II-§2.6.12) are written.

Parameters:

a- an array into which to place the results

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getRGB

public **final** **int** **getRGB**(**int** **pixel**)

Calculates a single integer representing the alpha, red, green, and blue components of the pixel in this color model. The components are each scaled to be a value between 0 and 255. The integer returned is the number such that bits 24-31 are the alpha value, 16-23 are the red value, bits 8-15 are the green value, and bits 0-7 are the blue value.

This format is the format used by the default RGB color model (II-§2.1.9).

Parameters:

pixel— a pixel value

Returns:

an integer representing this color in RGB format.

Overrides:

getRGB in class ColorModel (II-§2.1.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

IndexColorModel.getTransparentPixel

public final int getTransparentPixel()

Returns:

the index of the transparent pixel in this color model, or -1 if there is no transparent pixel.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.7 Class MemoryImageSource

```
public class java.awt.image.MemoryImageSource
    extends java.lang.Object (I-§1.12)
    implements java.awt.image.ImageProducer (II-§2.12)
{
    // Constructors
    public MemoryImageSource(int w, int h, ColorModel cm, §2.7.1
        byte pix[], int off, int scan);
    public MemoryImageSource(int w, int h, ColorModel cm, §2.7.2
        byte pix[], int off, int scan,
;        Hashtable props);
    public MemoryImageSource(int w, int h, ColorModel cm, §2.7.3
        int pix[], int off, int scan);
    public MemoryImageSource(int w, int h, ColorModel cm, §2.7.4
        int pix[], int off, int scan,
        Hashtable props);
    public MemoryImageSource(int w, int h, int pix[], §2.7.5
        int off, int scan);
    public MemoryImageSource(int w, int h, int pix[], §2.7.6
        int off, int scan,
        Hashtable props);

    // Methods
    public void addConsumer(ImageConsumer ic); §2.7.7
    public boolean isConsumer(ImageConsumer ic); §2.7.8
    public void removeConsumer(ImageConsumer ic); §2.7.9
    public void §2.7.10
        requestTopDownLeftRightResend(ImageConsumer ic);
    public void startProduction(ImageConsumer ic); §2.7.11
}
```

A memory image source is an implementation of the image producer interface (II-§2.12). It uses an array to produce pixel values for the image.

Here is an example which calculates a 100 × 100 image representing a fade from black to blue along the x axis and from black to red along the Y axis:

```
int w = 100;
int h = 100;
int pix[] = new int[w * h];
int index = 0;
for (int y = 0; y < h; y++) {
    int red = (y * 255) / (h - 1);
    for (int x = 0; x < w; x++) {
        int blue = (x * 255) / (w - 1);
        pix[index++] = (255 << 24) | (red << 16) | blue;
    }
}
```



```
Image img = createImage(new MemoryImageSource(w, h, pix, 0, w));
```

An application can use the method `createImage` in class `Component` (II-§1.10.6) to create an Image from a `MemoryImageSource`.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MemoryImageSource.MemoryImageSource

public

```
MemoryImageSource(int w, int h, ColorModel cm, byte pix[], int off,  
int scan)
```

Constructs an image producer ([II-§2.12](#)) object which uses an array of bytes to produce data for the image object. The pixel at coordinate (i, j) is stored in the pixel array at index $j \times scan + i + \text{offset}$.

Parameters:

- w- the width of the image
- h- the height of the image
- cm- the color model used by the pixels
- pix- the array of pixel values
- off- the offset of the first pixel in the array
- scan- the number of pixels in the array per line

```
public MemoryImageSource(int w, int h, ColorModel cm, byte pix[], int  
off, int scan, Hashtable props)
```

Constructs an image producer ([II-§2.12](#)) object which uses an array of bytes to produce data for the image object. The pixel at coordinate (i, j) is stored in the pixel array at index $j \times scan + i + \text{offset}$.

In addition, the image has the properties indicated in the hash table argument.

Parameters:

- w- the width of the image
- h- the height of the image
- cm- the color model used by the pixels
- pix- the array of pixel values
- off- the offset of the first pixel in the array

`scan`- the number of pixels in the array per line
`props`- a hash table of properties

```
public MemoryImageSource(int w, int h, ColorModel cm, int pix[], int  
off, int scan)
```

Constructs an image producer (II-§2.12) object which uses an array of integers to produce data for the image object. The pixel at coordinate (i, j) is stored in the pixel array at index $j \times \text{scan} + i + \text{offset}$.

Parameters:

`w`- the width of the image
`h`- the height of the image
`cm`- the color model used by the pixels
`pix`- the array of pixel values
`off`- the offset of the first pixel in the array
`scan`- the number of pixels in the array per line

```
public MemoryImageSource(int w, int h, ColorModel cm, int pix[], int  
off, int scan, Hashtable props)
```

Constructs an image producer (II-§2.12) object which uses an array of integers to produce data for the image object. The pixel at coordinate (i, j) is stored in the pixel array at index $j \times \text{scan} + i + \text{offset}$.

In addition, the image has the properties indicated in the hash table argument.

Parameters:

`w`- the width of the image
`h`- the height of the image
`cm`- the color model used by the pixels
`pix`- the array of pixel values
`off`- the offset of the first pixel in the array

`scan`- the number of pixels in the array per line
`props`- a hash table of properties

public **MemoryImageSource**(`int w, int h, int pix[], int off, int scan`)

Constructs an image producer (II-§2.12) object which uses an array of integers to produce data for the image object. The pixel at coordinate (`i, j`) is stored in the pixel array at index `j x scan + i + offset`.

The resulting image uses the default RGB color model (II-§2.1.9).

Parameters:

`w`- the width of the image
`h`- the height of the image
`pix`- the array of pixel values
`off`- the offset of the first pixel in the array
`scan`- the number of pixels in the array per line

public **MemoryImageSource**(`int w, int h, int pix[], int off, int scan, Hashtable props`)

Constructs an image producer (II-§2.12) object which uses an array of integers to produce data for the image object. The pixel at coordinate (`i, j`) is stored in the pixel array at index `j x scan + i + offset`. The resulting image uses the default RGB color model (II-§2.1.9).

In addition, the image has the properties indicated in the hash table argument.

Parameters:

`w`- the width of the image
`h`- the height of the image
`cm`- the color model used by the pixels
`pix`- the array of pixel values

`off`- the offset of the first pixel in the array

`scan`- the number of pixels in the array per line

`props`- a hash table of properties

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MemoryImageSource.AddConsumer

public void AddConsumer(ImageConsumer ic)

Registers the image consumer ([II-§2.10](#)) argument as wanting the image produced by this image producer. For more information on this method and its arguments, see the `AddConsumer` method ([II-§2.12.1](#)) of `ImageProducer`.

The `AddConsumer` method of `MemoryImageSource`, since it already has the image data, immediately delivers the data to the image consumer.

Parameters:

`ic`— an image consumer

{ewl msdncl.dll, ewcright, /c"Microsoft"}

MemoryImageSource.isConsumer

public **boolean** **isConsumer**(**ImageConsumer** **ic**)

Determines if the image consumer argument is registered with this image producer as a consumer. Because the memory image source delivers data immediately to its image consumer, the memory image source can have at most one consumer at a time.

Parameters:

ic— an image consumer

Returns:

true if the specified image consumer argument (II-§2.10) is currently registered with this image producer as one of its consumers; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MemoryImageSource.removeConsumer

public void removeConsumer(ImageConsumer ic)

Removes the specified image consumer ([II-§2.10](#)) object from the list of consumers registered to receive the image data. For more information on this method and its arguments, see the removeConsumer method ([II-§2.12.3](#)) of ImageProducer.

Parameters:

ic- an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MemoryImageSource.requestTopDownLeftRightResend

public void requestTopDownLeftRightResend(ImageConsumer ic)

An image consumer sends this message to request that the image producer attempt to resend the image data one more time in top-down, left-to-right order. For more information on this method and its arguments, see the requestTopDownLeftRightResend method (II-§2.12.4) of ImageProducer.

The requestTopDownLeftRightResend method of memoryImageSource ignores this request, since it always sends its data in that order.

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MemoryImageSource.startProduction

public void startProduction(ImageConsumer ic)

Registers the image consumer ([II-§2.10](#)) argument as wanting the image produced by this image producer. In addition, this method forces the image producer to start an immediate reconstruction of the image data. For more information on this method and its arguments, see the startProduction method ([II-§2.12.5](#)) of ImageProducer.

The addConsumer method of MemoryImageSource, since it already has the image data, immediately delivers the data to the image consumer.

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.8 Class PixelGrabber

```
public class java.awt.image.PixelGrabber
    extends java.lang.Object (I-§1.12)
    implements java.awt.image.ImageConsumer (II-§2.10)
{
    // Constructors
    public PixelGrabber(Image img, int x, int y, §2.8.1
        int w, int h, int pix[], int off, int scansize);
    public PixelGrabber(ImageProducer ip, int x, §2.8.2
        int y, int w, int h, int pix[], int off, int scansize);

    // Methods
    public boolean grabPixels(); §2.8.3
    public boolean grabPixels(long ms); §2.8.4
    public void imageComplete(int status); §2.8.5
    public void setColorModel(ColorModel model); §2.8.6
    public void setDimensions(int width, int height) §2.8.7
        public void setHints(int hints); §2.8.8
    public void setPixels(int srcX, int srcY, int srcW, §2.8.9
        int srcH, ColorModel model,
        byte pixels[], int srcOff, int srcScan);
    public void setPixels(int srcX, int srcY, int srcW, §2.8.10
        int srcH, ColorModel model,
        int pixels[], int srcOff, int srcScan);
    public void setProperties(Hashtable props); §2.8.11
    public int status(); §2.8.12
}
```

The pixel grabber implements an image consumer which can be attached to an image or image producer object to retrieve a subset of the pixels in that image. Pixels are stored in the array in the default RGB color model (II-§2.1.9).

For example:

```
public void handleSinglePixel(int x, int y, int pixel) {
    int alpha = (pixel >> 24) & 0xff;
    int red    = (pixel >> 16) & 0xff;
    int green  = (pixel >>  8) & 0xff;
    int blue   = (pixel          ) & 0xff;
    // Deal with the pixel as necessary...
}

public void GetPixels(Image img, int x, int y, int w, int h) {
    int[] pixels = new int[w * h];
    PixelGrabber pg =
        new PixelGrabber(img, x, y, w, h, pixels, 0, w);
    try {
```



```

        pg.grabPixels();
    } catch (InterruptedException e) {
        System.err.println("interrupted waiting for pixels!");
        return;
    }
    if ((pg.status() & ImageObserver.ABORT) != 0) {
        System.err.println("image fetch aborted or errored");
        return;
    }
    for (int j = 0; j < h; j++) {
        for (int i = 0; i < w; i++) {
            // look at the pixel
            handleSinglePixel(x+i, y+j, pixels[j * w + i]);
        }
    }
}

```

Most applications need to call only the grabPixel methods ([§2.8.3](#), [§2.8.4](#)) and the status method ([II-§2.8.12](#)) of this class. The remaining methods are part of the ImageConsumer interface and allow the pixel grabber to receive the image from the image producer.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PixelGrabber.PixelGrabber

```
public PixelGrabber(Image img, int x, int y, int w, int h, int pix[],  
int off, int scansize)
```

Creates a new pixel grabber object to grab the rectangular section of pixels from the specified image into the specified array.

The pixels are stored in the array in the default RGB color model (II-§2.1.9). The pixel data for the coordinate (i, j) , where (i, j) is inside the indicted rectangle, is stored in the array at index

$$(j - y) \times \text{scan} + (i - x + \text{offset})$$

of the pixel array.

The x and y coordinates indicate the upper-left corner of the rectangle of pixels to retrieve from the image, relative to the default (unscaled) size of the image.

Parameters:

img- the image from which to retrieve pixels

x- the x coordinate of the upper-left corner

y- the y coordinate of the upper-left corner

w- the width of the rectangle to retrieve

h- the height of the rectangle to retrieve

pix- the array of integers into which to place the RGB pixels retrieved from the image

off- the offset into the array to store the first pixel

scansize- the distance from the start of one row of pixels to the start of the next row in the array

```
public PixelGrabber(ImageProducer ip, int x, int y, int w, int h, int  
pix[], int off, int scansize)
```


Creates a new pixel grabber object to grab the rectangular section of pixels from the specified image producer into the specified array.

The pixels are stored in the array in the default RGB color model (II-§2.1.9). The pixel data for the coordinate (i, j) , where (i, j) is inside the indicted rectangle, is stored in the array at the index

$$(j - y) \times \text{scan} + (i - x) + \text{offset}$$

of the pixel array.

The x and y coordinates indicate the upper-left corner of the rectangle of pixels to retrieve from the image, relative to the default (unscaled) size of the image.

Parameters:

ip- the image producer

x- the x coordinate of the upper-left corner

y- the y coordinate of the upper-left corner

w- the width of the rectangle to retrieve

h- the height of the rectangle to retrieve

pix- the array of integers into which to place the RGB pixels retrieved from the image

off- the offset into the array to store the first pixel

scansize- the distance from the start of one row of pixels to the start of the next row in the array

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


PixelGrabber.grabPixels

public boolean grabPixels()
throws InterruptedException

Requests the image or image producer to start delivering pixels to this image consumer. It waits for all of the pixels in the rectangle of interest to be delivered.

Returns:

true if the pixels were successfully grabbed; false on abort or error.

Throws

InterruptedException ([I-§1.37](#))

If another thread has interrupted this thread.

public boolean grabPixels(long ms)
throws InterruptedException

Requests the image or image producer to start delivering pixels to this image consumer. It waits for all of the pixels in the rectangle of interest to be delivered, or until the specified timeout has elapsed.

Parameters:

ms – the number of milliseconds to wait for the pixels

Returns:

true if the pixels were successfully grabbed; false on abort, error, or timeout.

Throws

InterruptedException ([I-§1.37](#))

If another thread has interrupted this thread.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.imageComplete

public void imageComplete(int status)

The image producer calls the imageComplete method when it has completed an image or it has errored in producing or loading the image. For more information on this method and its status argument, see §2.10.10.

The imageComplete method of PixelGrabber notifies all processes waiting for the pixels to wake up. It uses the value of the status flag to determine whether the image was successfully retrieved or not.

Parameters:

status- the status of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.setColorModel

public void setColorModel (ColorModel model)

The image producer calls the setColorModel method to specify the color model for the majority of the subsequent setPixels method calls. For more information on this method and its model argument, see §2.10.11.

The setColorModel method of PixelGrabber ignores this call.

Parameters:

model- a color map used in subsequent setPixel calls

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.setDimensions

public void setDimensions(int width, int height)

The image producer calls the setDimensions of the image consumer to tell it the width and height of the image.

The setDimensions method of PixelGrabber ignores the dimensions.

Parameters:

width- the width of the image

height- the height of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.setHints

public void setHints(int hints)

The image producer calls the setHints method of the image consumer to indicate the order in which the bits will be delivered. For more information on the hints passed to the image consumer, see §2.10.13.

The setHints method of PixelGrabber ignores the hints.

Parameters:

hints- hints about the order in which the bits will be delivered

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.setPixels

```
public void setPixels(int srcX, int srcY, int srcW, int srcH,  
ColorModel model, byte pixels[],int srcOff, int srcScan)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.14.

The setPixels method of PixelGrabber places the bits, if appropriate, into the array of bits passed to it by a call to the grabPixels method (II-§2.8.3).

Parameters:

x- left coordinate of rectangle
y- top coordinte of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

```
public void
```

```
setPixels(int srcX, int srcY, int srcW, int srcH, ColorModel model,  
int pixels[],int srcOff, int srcScan)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.15.

The setPixels method of PixelGrabber places the bits, if appropriate, into the array of bits passed to it by a call to the grabPixels method (II-§2.8.3).

Parameters:

x- left coordinate of rectangle

y- top coordinte of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.setProperties

public void setProperties (Hashtable props)

The image producer calls the setProperties method of the image consumer to let it know of additional properties of the image. For more information on this method and its arguments, see §2.10.16.

The setPropertyess method of PixelGrabber ignores the hints.

Parameters:

props- a hash table that maps image properties to their value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

PixelGrabber.status

public int status()

Returns the bitwise OR of the appropriate ImageObserver interface (II-§2.11) flags.

Returns:

the status of the pixels.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.9 Class RGBImageFilter

```
public abstract class java.awt.image.RGBImageFilter
    extends java.awt.image.ImageFilter (II-§2.5)
{
    // Fields
    protected boolean canFilterIndexColorModel; §2.9.1
    protected ColorModel newmodel; §2.9.2
    protected ColorModel origmodel; §2.9.3

    // Constructors
    public RGBImageFilter(); §2.9.4

    // Methods
    public IndexColorModel §2.9.5
        filterIndexColorModel(IndexColorModel icm);
    public abstract int filterRGB(int x, int y, int rgb); §2.9.6
    public void filterRGBPixels(int x, int y, int w, §2.9.7
        int h, int pixels[], int off, int scansize);
    public void setColorModel(ColorModel model); §2.9.8
    public void setPixels(int x, int y, int w, int h, §2.9.9
        ColorModel model, byte pixels[], int off, int scansize);
    public void setPixels(int x, int y, int w, int h §2.9.10
        ColorModel model, int pixels[], int off, int scansize);
    public void tstituteColorModel(ColorModel oldcm, §2.9.11
        ColorModel newcm);
}
```

This class provides an easy way to create an image filter (II-§2.5) which modifies the pixels of the original image by converting them one at a time in the default RGB color model (II-§2.1.9).

Objects of this class are meant to be used in conjunction with a filtered image source (II-§2.4) object to produce filtered versions of existing images.

This class is an abstract class. It provides the calls needed to channel all the pixel data through a single method which converts pixels, one at a time, into the default RGB color model, regardless of the color model being used by the image producer. The only method which needs to be defined to create a useable image filter is the filterRGB method.

Following is an example of a filter which swaps the red and blue components of an image:

```
class RedBlueSwapFilter extends RGBImageFilter {
    public RedBlueSwapFilter() {
        // The filter's operation does not depend on the
        // pixel's location, so IndexColorModels can be
        // filtered directly.
        canFilterIndexColorModel = true;
    }
    public int filterRGB(int x, int y, int rgb) {
```



```
        return ((rgb & 0xff00ff00)
                | ((rgb & 0xff0000) >> 16)
                | ((rgb & 0xff) << 16));
    }
```

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RGBImageFilter.canFilterIndexColorModel

protected **boolean** canFilterIndexColorModel

Setting this value to true indicates that the value returned by the filterRGB method (II-§2.9.6) is independent of the x and y arguments, and depends only on the rgb argument.

Subclasses of RGBImageFilter should set this field to true in their constructor if their filterRGB method does not depend on the coordinate of the pixel being filtered. Filtering the colormap entries of an indexed color map can be much faster than filtering every pixel.

The default value is false.

See Also:

substituteColorModel (II-§2.9.11)

IndexColorModel (II-§2.6).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.newmodel

protected ColorModel newmodel

This field is used to remember the newcm argument passed to the substituteColorModel (II-§2.9.11) method.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.origmodel

protected ColorModel origmodel

This field is used to remember the oldcm argument passed to the substituteColorModel (II-§2.9.11) method.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.RGBImageFilter

public RGBImageFilter()

The default constructor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.filterIndexColorModel

public `IndexColorModel`

`filterIndexColorModel(IndexColorModel icm)`

Filters an index color model ([II-§2.6](#)) object by running each entry in its color table through the `filterRGB` method ([II-§2.9.6](#)). The call to `filterRGB` has the `x` and `y` arguments set to -1 as a flag to indicate that a color table entry is being filtered rather than an actual pixel value.

Parameters:

`icm`— the index color model object to be filtered

Returns:

a new index color model with the filtered colors.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.filterRGB

`public abstract int filterRGB(int x, int y, int rgb)`

Specifies a method to convert a single input pixel, whose value is specified in the default RGB color model (II-§2.1.9), to a new pixel value also in the default RGB color model. Subclasses of RGBImageFilter must provide a definition for this method.

If the value of the field canFilterIndexColorModel (II-§2.9.1) is true, then the value returned by this method must not depend on the x and y coordinates.

If the x and y arguments are both -1, this method is being called by the filterIndexColorModel method (II-§2.9.5).

Parameters:

- x- the x coordinate of the pixel
- y- the y coordinate of the pixel
- rgb- the value of the pixel, in the default RGB color model

Returns:

the new value of the pixel, in the default RGB color model.

See Also:

filterRGBPixels (II-§2.9.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.filterRGBPixels

```
public void filterRGBPixels(int x, int y, int w, int h, int pixels[],  
int off, int scansize)
```

Filters a buffer of pixels in the default RGB color model (II-§2.1.9) by passing them one by one through the filterRGB method (II-§2.9.6).

The setPixels method (II-§2.10.15) of the filter's consumer (II-§2.5.1) is then called with the resulting buffer and the color model argument set to the default RGB color model.

Only pixels that fall within the specified rectangle are modified. The value of the pixel at coordinate (i, j) is stored in the pixel array at index $j \times \text{scan} + i + \text{offset}$.

Parameters:

x- left coordinate of rectangle
y- top coordinate of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RGBImageFilter.setColorModel

public void setColorModel (ColorModel model)

The image producer calls the setColorModel method to specify the color model for the majority of the subsequent setPixels method calls. For more information on this method and its model argument, see §2.10.11

The setColorModel method of RGBImageFilter determines if the color model argument is an index color model (II-§2.6) and if the canFilterIndexColorModel field (II-§2.9.1) is true.

If both conditions are true, the method creates a new color model by calling the filterIndexColorModel method (II-§2.9.5) on the model argument. The original color model and the newly created color model are then passed as arguments to the substituteColorModel (II-§2.9.11) method. In addition, the setColorModel method (II-§2.9.8) of the filter's consumer (II-§2.5.1) is called with the newly created color model.

If either condition is false, the method calls the the setColorModel method (II-§2.10.11) of its consumer (II-§2.5.1) with the default RGB color map (II-§2.1.9).

Parameters:

`model` – a color map used in subsequent setPixel calls

Overrides:

setColorModel in class ImageFilter (II-§2.5.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.setPixels

```
public void setPixels(int x, int y, int w, int h, ColorModel model,  
byte pixels[], int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.14.

The setPixels method of RGBImageFilter looks to see if the color model is the same one that has already been converted and remembered for substitution by a previous call to the substituteColorModel (II-§2.9.11) method.

If so, it calls the setPixels method (II-§2.10.14) of its consumer (II-§2.5.1), changing the color model argument to be the alternative color model.

Otherwise, the method converts the buffer of byte pixels to the default RGB color model (II-§2.1.9) and passes the converted buffer to the filterRGBPixels (II-§2.9.7) method to be converted one by one.

Parameters:

x- left coordinate of rectangle
y- top coordinate of rectangle
w- width of rectangle
h- height of rectangle
model- color model for bits
pixels- array of bits
off- offset for first element
scansize- number of elements per row

Overrides:

setPixels in class ImageFilter (II-§2.5.10).

```
public void setPixels(int x, int y, int w, int h, ColorModel model, int  
pixels[], int off, int scansize)
```


The image producer calls the `setPixels` method of the image consumer one or more times to deliver the pixels of the image. For more information on this method and its arguments, see §2.10.15.

The `setPixels` method of `RGBImageFilter` looks to see if the color model is the same one that has already been converted and remembered for substitution by a previous call to the `substituteColorModel` (II-§2.9.11) method.

If so, it calls the `setPixels` method (II-§2.10.14) of the filter's consumer (II-§2.5.1), changing the color model argument to be the alternative color model.

Otherwise, the method converts the buffer of byte pixels to the default RGB color model (II-§2.1.9) and passes the converted buffer to the `filterRGBPixels` (II-§2.9.7) method to be converted one by one.

Parameters:

`x`- left coordinate of rectangle
`y`- top coordinate of rectangle
`w`- width of rectangle
`h`- height of rectangle
`model`- color model for bits
`pixels`- array of bits
`off`- offset for first element
`scansize`- number of elements per row

Overrides:

`setPixels` in class `ImageFilter` (II-§2.5.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

RGBImageFilter.substituteColorModel

public void substituteColorModel(ColorModel oldcm, ColorModel newcm)

Registers two color model objects for substitution. If the oldcm is the color model during any subsequent call to either of the setPixels methods ([§2.9.9](#), [§2.9.10](#)), the newcm argument is substituted and the pixels passed through unmodified.

Parameters:

oldcm- the ColorModel object to be replaced on the fly

newcm- the ColorModel object to replace oldcm on the fly

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.10 Interface ImageConsumer

```
public interface java.awt.image.ImageConsumer
{
    // status value for the imageComplete method
    public final static int IMAGEABORTED;    §2.10.1
    public final static int IMAGEERROR;    §2.10.2
    public final static int SINGLEFRAMEDONE;    §2.10.3
    public final static int STATICIMAGEDONE;    §2.10.4

    // hints used by the setHints method
    public final static int COMPLETESCANLINES;    §2.10.5
    public final static int RANDOMPIXELORDER;    §2.10.6
    public final static int SINGLEFRAME;    §2.10.7
    public final static int SINGLEPASS;    §2.10.8
    public final static int TOPDOWNLEFTRIGHT;    §2.10.9

    // Methods
    public abstract void imageComplete(int status);    §2.10.10
    public abstract void setColorModel(ColorModel model);    §2.10.11
    public abstract void §2.10.12
        setDimensions(int width, int height);
    public abstract void setHints(int hintflags);    §2.10.13
    public abstract void §2.10.14
        setPixels(int x, int y, int w, int h,
                    ColorModel model, byte pixels[],
                    int off, int scansize);
    public abstract void §2.10.15
        setPixels(int x, int y, int w, int h,
                    ColorModel model, int pixels[],
                    int off, int scansize);
    public abstract void setProperties(Hashtable props);    §2.10.16
}
```

The image consumer interface specifies the methods that all image consumers must implement. An image consumer is an object interested in data produced by the image producers (II-§2.12).

When a consumer is added to an image producer, the producer delivers all the data about the image using the method calls defined in this interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.IMAGEABORTED

```
public final static int IMAGEABORTED = 4
```

Argument to the `imageComplete` method (II-§2.10.10) indicating that the image creation process was deliberately aborted.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.IMAGEERROR

```
public final static int IMAGEERROR = 1
```

Argument to the imageComplete method (II-§2.10.10) indicating that an error was encountered while producing the image.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.SINGLEFRAMEDONE

```
public final static int SINGLEFRAMEDONE = 2
```

Argument to the `imageComplete` [method \(II-§2.10.10\)](#) indicating that one frame of the image is complete but there are more frames to be delivered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.STATICIMAGEDONE

```
public final static int STATICIMAGEDONE = 3
```

Argument to the imageComplete method (II-§2.10.10) indicating that the image is complete and there are no more pixels or frames to be delivered.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.COMPLETESCANLINES

```
public final static int COMPLETESCANLINES = 4
```

Flag in the setHints method (II-§2.10.13) indicating that the pixels will be delivered in (multiples of) complete scanlines at a time.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.RANDOMPIXELORDER

```
public final static int RANDOMPIXELORDER = 1
```

Flag in the setHints method (II-§2.10.13) indicating that the pixels will be delivered in a random order.

The image consumer should not use any optimizations that depend on the order of pixel delivery. If the image producer doesn't call the setHints method, the image consumer assumes that the bits are being delivered in a random order.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.SINGLEFRAME

`public final static int SINGLEFRAME = 8`

Flag in the `setHints` [method \(II-§2.10.13\)](#) indicating that the image contains a single static image. The `pixels` will be defined in calls to the `setPixels` methods; the image producer then calls the `imageComplete` [method \(II-§2.10.10\)](#) with the `STATICIMAGEDONE` [flag \(II-§2.10.4\)](#) after which no more image data is delivered.

Examples of image types that do not meet these criteria include the output of a video feed, or the representation of a 3D rendering being manipulated by the user. The end of each frame of those types of images is indicated by the image producer calling the `imageComplete` [method \(II-§2.10.10\)](#) with the `SINGLEFRAMEDONE` [flag \(II-§2.10.4\)](#).

`{ewl msdncd.dll, ewcright, /c"Microsoft"}`

ImageConsumer.SINGLEPASS

```
public final static int SINGLEPASS = 8
```

Flag in the setHints method (II-§2.10.13) indicating that each pixel will be delivered only once.

An image type that does not meet this criterion is a progressive JPEG image which defines pixels in multiple passes, each more refined than the previous.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageConsumer.TOPDOWNLEFTTRIGH

public final static int TOPDOWNLEFTRIGHT

Flag in the setHints method (II-§2.10.13) indicating that the pixels will be delivered in a top-down, left-to-right order.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.imageComplete

public abstract void imageComplete(int status)

The image producer calls the imageComplete method when one of the following conditions has occurred:

- it has delivered all the pixels that the source image contains
- a single frame of a multi-frame animation has been completed
- an error in loading or producing the image has occurred
- the image production was explicitly aborted by the application

The image consumer should remove itself from the list of consumers registered with the image producer (II-§2.12.3), unless it is interested in subsequent frames.

The status is one of the following values:

(II-§2.10.2)(II-§2.10.7)(II-§2.10.4)(II-§2.10.1)

Parameters:

status- the status of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.setColorModel

public abstract void setColorModel(ColorModel model)

The image producer calls the setColorModel method to specify the color model (II-§2.1) for the majority of the subsequent setPixels method calls.

Each set of pixels delivered using the setPixels method includes its own color model, so the image consumer should not assume that the model argument is the color model argument in every subsequent setPixels method call.

A notable case where multiple ColorModel objects may be seen is a filtered image where for each set of pixels that it filters, the filter determines whether the pixels can be sent on untouched, using the original ColorModel, or should be modified (filtered) and passed on using a ColorModel more convenient for the filtering process.

Parameters:

model— a color map used in subsequent setPixel calls

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.setDimensions

public abstract void

setDimensions(int width, int height)

The image producer calls the setDimensions of the image consumer to indicate the width and height of the image.

Parameters:

width- the width of the image

height- the height of the image

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.setHints

public abstract void setHints(int hintflags)

The image producer calls the setHints method of the image consumer to indicate the order in which the bits will be delivered.

The image producer is allowed to deliver the pixels in any order, but the image consumer may be able to scale or convert the pixels more efficiently or with higher quality if it knows some information about how the pixels will be presented.

The image producer should call the setHints method before any calls to the image consumer's setPixels method.

The hintflags argument is a bit mask of hints about the manner in which the pixels are delivered.

The possible hint flags are:

(II-§2.10.5)(II-§2.10.5)(II-§2.10.5)(II-§2.10.5)(II-§2.10.5)

Parameters:

hints- hints about the order in which the bits will be delivered

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.setPixels

public abstract void

```
setPixels(int x, int y, int w, int h, ColorModel model, byte  
pixels[], int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. Each call specifies the location and size of the rectangle of source pixels contained in the array of pixels.

The specified color model object should be used to convert the pixels into their corresponding color and alpha components. The pixel at coordinate (i, j) is stored in the pixel array at index

$$(j - y) \times \text{scan} + (i - x) + \text{offset}$$

The pixels delivered using this method are all stored as bytes.

Parameters:

x- left coordinate of rectangle

y- top coordinate of rectangle

w- width of rectangle

h- height of rectangle

model- color model for bits

pixels- array of bits

off- offset for first element

scansize- number of elements per row

public abstract void

```
setPixels(int x, int y, int w, int h, ColorModel model, int pixels[],  
int off, int scansize)
```

The image producer calls the setPixels method of the image consumer one or more times to deliver the pixels of the image. Each call specifies the location and size of the rectangle of

source pixels that are contained in the array of pixels.

The specified color model object should be used to convert the pixels into their corresponding color and alpha components. The pixel at coordinate {ewc msdncd, EWGraphic, IMA14ga 0 /a "sunref.BMP"} is stored in the pixel array at index

{ewc msdncd, EWGraphic, IMA14ga 1 /a "sunref.BMP"}

The pixels delivered using this method are all stored as integers.

Parameters:

x- left coordinate of rectangle

y- top coordinate of rectangle

w- width of rectangle

h- height of rectangle

model- color model for bits

pixels- array of bits

off- offset for first element

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageConsumer.setProperties

public abstract void setProperties (Hashtable props)

The image producer calls the setProperties method of the image consumer to indicate additional properties of the image.

All keys to the hash table are strings. The corresponding values depend on the string.

Parameters:

props- a hash table of properties

See Also:

getProperty in class Image ([II-§1.24.6](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.11 Interface ImageObserver

```
public interface java.awt.image.ImageObserver
{
    // flags for the infoflags argument to imageUpdate
    public final static int ABORT; §2.11.1
    public final static int ALLBITS; §2.11.2
    public final static int ERROR; §2.11.3
    public final static int FRAMEBITS; §2.11.4
    public final static int HEIGHT; §2.11.5
    public final static int PROPERTIES; §2.11.6
    public final static int SOMEBITS; §2.11.7
    public final static int WIDTH; §2.11.8

    // Methods
    public abstract boolean §2.11.9
        imageUpdate(Image img, int infoflags,
            int x, int y, int width, int height);
}
```

The image observer interface specifies the methods that all image observers must implement.

An image observer is interested in receiving asynchronous notifications about the image as the image is being constructed.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.ABORT

```
public final static int ABORT = 128
```

This flag in the infoflags argument to `imageUpdate` ([II-§2.11.9](#)) indicates that the image was aborted before production was complete.

No more information will become available without further action to trigger another image production sequence.

If the `ERROR` flag was not also set in this image update, then accessing any of the data in the image restarts the production again, possibly from the beginning.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.ALLBITS

```
public final static int ALLBITS = 32
```

This flag in the infoflags argument to `imageUpdate` ([II-§2.11.9](#)) indicates that a static image is now complete and can be drawn in its final form.

The `x`, `y`, `width`, and `height` arguments to the `imageUpdate` method should be ignored when this flag is set in the status.

See Also:

`drawImage` in class `Graphics` ([II-§1.20.12](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.ERROR

```
public final static int ERROR = 64
```

This flag in the infoflags argument to `imageUpdate` ([II-§2.11.9](#)) indicates that an image which was being tracked asynchronously has encountered an error. No further information will become available, and drawing the image will fail.

Whenever this flag is set, the ABORT flag must also be set.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.FRAMEBITS

`public final static int FRAMEBITS = 16`

This flag in the infoflags argument to `imageUpdate` ([II-§2.11.9](#)) indicates that another complete frame of a multi-frame image can now be drawn.

The x, y, width, and height arguments to the `imageUpdate` method should be ignored when this flag is set in the status.

See Also:

`drawImage` in class `Graphics` ([II-§1.20.12](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageObserver.HEIGHT

```
public final static int HEIGHT = 2
```

This flag in the infoflags argument to `imageUpdate` ([II-§2.11.9](#)) indicates that the height of the base image is now available and can be taken from the height argument to the `imageUpdate` method.

See Also:

`getHeight` in class `Image` ([II-§1.24.5](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.PROPERTIES

```
public final static int PROPERTIES = 4
```

This flag in the infoflags argument to imageUpdate (II-§2.11.9) indicates that the properties of the image are now available.

See Also:

getProperty in class Image (II-§1.24.6).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.SOMEBITS

```
public final static int SOMEBITS = 8
```

This flag in the infoflags argument to imageUpdate ([II-§2.11.9](#)) indicates that the pixels needed for drawing a scaled variation of the image are now available.

The bounding box of the new pixels can be taken from the x, y, width, and height arguments to the imageUpdate method.

See Also:

drawImage in class Graphics ([II-§1.20.12](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageObserver.WIDTH

public final static int WIDTH

This flag in the infoflags argument to imageUpdate (II-§2.11.9) indicates that the width of the base image is now available and can be taken from the width argument to the imageUpdate method.

See Also:

getWidth in class Image (II-§1.24.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageObserver.imageUpdate

public abstract boolean

imageUpdate(Image img, int infoflags, int x, int y, int width, int height)

This image observer method is called when previously requested information about an image becomes available.

Asynchronous interfaces are method calls such as getWidth (II-§1.24.8), getHeight (II-§1.24.5), and drawImage (II-§1.20.12) which takes an image observer as an argument. These methods register the caller as being interested either in information about the image or about an output version of the image.

This method should return true if further calls to imageUpdate are needed by this image observer; false if it needs no more information.

The infoflags argument should be the OR of the following flags:

(II-§2.11.8)(II-§2.11.5)(II-§2.11.6)(II-§2.11.7)(II-§2.11.4)(II-§2.11.2)(II-§2.11.3)(II-§2.11.1)

The interpretation of the x, y, width, and height arguments depends on the infoflags argument.

Parameters:

img- The image being observed
infoflags- the OR of the above flags
x- an x coordinate
y- a y coordinate
width- the width
height- the height

Returns:

true if further calls to imageUpdate are needed by this image observer; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§2.12 Interface ImageProducer

```
public interface java.awt.image.ImageProducer
{
    // Methods
    public abstract void addConsumer(ImageConsumer ic); §2.12.1
    public abstract boolean isConsumer(ImageConsumer ic); §2.12.2
    public abstract void removeConsumer(ImageConsumer ic); §2.12.3
    public abstract void §2.12.4
        requestTopDownLeftRightResend(ImageConsumer ic);
    public abstract void startProduction(ImageConsumer ic); §2.12.5
}
```

The image producer interface specifies the methods that all image producers must implement. Every image contains an image producer which can reconstruct the image whenever it is needed by an image consumer (II-§2.10).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageProducer.addConsumer

`public abstract void addConsumer(ImageConsumer ic)`

Registers the image consumer argument as wanting information about this image.

The image producer may, at its discretion, start delivering the image data immediately, or it may wait until the next image reconstruction is forced by a call to the startProduction method (II-§2.12.5).

Parameters:

`ic`— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageProducer.isConsumer

public abstract boolean isConsumer(ImageConsumer ic)

Parameters:

ic- an image consumer

Returns:

true if the specified image consumer argument is currently registered with this image producer as one of its consumers; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageProducer.removeConsumer

public abstract void removeConsumer(ImageConsumer ic)

Removes the specified image consumer object from the list of consumers registered to receive the image data. It is not an error to remove a consumer that is not registered.

The image producer should stop sending data to this consumer as soon as it is feasible.

Parameters:

ic— an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ImageProducer.requestTopDownLeftRightResend

public abstract void

requestTopDownLeftRightResend(ImageConsumer ic)

An image consumer invokes this method to request that the image producer attempt to resend the image data one more time in top-down, left-to-right order.

If the data cannot be resent in that order, the image producer ignores this call.

If the data can be resent in that order, the image producer should respond by executing the following minimum set of image consumer method calls:

```
ic.setHints(TOPDOWNLEFTRIGHT | otherhints );  
ic.setPixels(...); // As many times as needed  
ic.imageComplete();
```

An image consumer might call this method so that it can use a higher quality conversion algorithm which depends on receiving the pixels in order.

Parameters:

ic- an image consumer

See Also:

setHints in class ImageConsumer ([II-§2.10.13](#)).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageProducer.startProduction

public abstract void startProduction(ImageConsumer ic)

Registers the image consumer argument as wanting information about this image.

In addition, this method forces the image producer to start an immediate reconstruction of the image data. The data will be delivered both to this image consumer and to any other image consumers which may have already been registered with the producer using the addConsumer method (II-§2.12.1)

Parameters:

ic- an image consumer

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.awt.peer

Interfaces

- [3.1 Interface ButtonPeer](#)
- [3.2 Interface CanvasPeer](#)
- [3.3 Interface CheckboxMenuItemPeer](#)
- [3.4 Interface CheckboxPeer](#)
- [3.5 Interface ChoicePeer](#)
- [3.6 Interface ComponentPeer](#)
- [3.7 Interface ContainerPeer](#)
- [3.8 Interface DialogPeer](#)
- [3.9 Interface FileDialogPeer](#)
- [3.10 Interface FramePeer](#)
- [3.11 Interface LabelPeer](#)
- [3.12 Interface ListPeer](#)
- [3.13 Interface MenuBarPeer](#)
- [3.14 Interface MenuComponentPeer](#)
- [3.15 Interface MenuItemPeer](#)
- [3.16 Interface MenuPeer](#)
- [3.17 Interface PanelPeer](#)
- [3.18 Interface ScrollbarPeer](#)
- [3.19 Interface TextAreaPeer](#)
- [3.20 Interface TextComponentPeer](#)
- [3.21 Interface TextFieldPeer](#)
- [3.22 Interface WindowPeer](#)

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.1 Interface ButtonPeer

```
public interface java.awt.peer.ButtonPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void setLabel(String label); §3.1.1
}
```

The button peer interface specifies the methods that all implementations of Abstract Window Toolkit buttons must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ButtonPeer.setLabel

public abstract void setLabel(String label)

Changes the button's label to be the String argument.

Parameters:

label- the new label, or null for no label

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.2 Interface CanvasPeer

```
public interface java.awt.peer.CanvasPeer  
    extends java.awt.peer.ComponentPeer (II-§3.6)  
{  
}
```

The canvas peer interface specifies the methods that all implementations of Abstract Window Toolkit canvases must define.

In general, canvas implementations only need to implement methods required of all component peers.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.3 Interface CheckboxMenuItemPeer

```
public interface java.awt.peer.CheckboxMenuItemPeer
    extends java.awt.peer.MenuItemPeer (II-§3.15)
{
    // Methods
    public abstract void setState(boolean t); §3.3.1
}
```

The check box menu peer interface specifies the methods that all implementations of Abstract Window Toolkit check box menus must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CheckboxMenuItemPeer.setState

public abstract void setState(boolean t)

Sets the check box to the specified boolean state: true indicates "on"; false indicates "off."

Parameters:

state- the boolean state

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.4 Interface CheckboxPeer

```
public interface java.awt.peer.CheckboxPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void setCheckboxGroup(CheckboxGroup g); §3.4.1
    public abstract void setLabel(String label); §3.4.2
    public abstract void setState(boolean state); §3.4.3
}
```

The check box peer interface specifies the methods that all implementations of Abstract Window Toolkit check boxes must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CheckboxPeer.setCheckboxGroup

public abstract void setCheckboxGroup (CheckboxGroup g)

Sets the group of the checkbox to be the specified CheckboxGroup

Parameters:

g- the new check box group, or null to remove the check box from any checkbox group

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxPeer.setLabel

public abstract void setLabel(String label)

Changes the check box's label to be the string argument.

Parameters:

label- the new label, or null for no label

{ewl msdncd.dll, ewcright, /c"Microsoft"}

CheckboxPeer.setState

public abstract void setState(boolean state)

Sets the check box to the specified boolean state: true indicates "on"; false indicates "off."

Parameters:

state- the boolean state

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.5 Interface ChoicePeer

```
public interface java.awt.peer.ChoicePeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void addItem(String item, int index); §3.5.1
    public abstract void select(int index); §3.5.2
}
```

The choice menu peer interface specifies the methods that all implementations of Abstract Window Toolkit choice menus must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ChoicePeer.addItem

public abstract void addItem(String item, int index)

Adds an item to the choice menu at the specified position.

Parameters:

item- the string item

index- the position

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ChoicePeer.select

public abstract void select(int index)

Sets the selected item to be the item at the specified position.

Parameters:

index- the selected item position

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.6 Interface ComponentPeer

```
public interface java.awt.peer.ComponentPeer
{
    // Methods
    public abstract int §3.6.1
        checkImage(Image img, int w, int h, ImageObserver o);
    public abstract Image §3.6.2
        createImage(ImageProducer producer);
    public abstract Image §3.6.3
        createImage(int width, int height);
    public abstract void disable(); §3.6.4
    public abstract void dispose(); §3.6.5
    public abstract void enable(); §3.6.6
    public abstract ColorModel getColorModel(); §3.6.7
    public abstract FontMetrics getFontMetrics(Font font); §3.6.8
    public abstract Graphics getGraphics(); §3.6.9
    public abstract Toolkit getToolkit(); §3.6.10
    public abstract boolean handleEvent(Event e); §3.6.11
    public abstract void hide(); §3.6.12
    public abstract Dimension minimumSize(); §3.6.13
    public abstract void nextFocus(); §3.6.14
    public abstract void paint(Graphics g); §3.6.15
    public abstract Dimension preferredSize(); §3.6.16
    public abstract boolean §3.6.17
        prepareImage(Image img, int w, int h, ImageObserver o);
    public abstract void print(Graphics g); §3.6.18
    public abstract void repaint(long tm, int x, int y, §3.6.19
        int width, int height);
    public abstract void requestFocus(); §3.6.21
    public abstract void reshape(int x, int y, int width, §3.6.22
        int height);
    public abstract void setBackground(Color c); §3.6.23
    public abstract void setFont(Font f); §3.6.24
    public abstract void setForeground(Color c); §3.6.25
    public abstract void show(); §3.6.26
}
```

The component peer interface specifies the methods that all implementations of Abstract Window Toolkit components must define.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.checkImage

public abstract int

checkImage(Image img, int w, int h, ImageObserver o)

Returns the status of the construction of a scaled screen representation of the specified image.

This method does not cause the image to begin loading. An application must use the prepareImage ([II-§3.6.17](#)) method to force the loading of an image.

Information on the flags returned by this method can be found in ([II-§2.11](#)).

Parameters:

- img- the image whose status is being checked
- w- the width of the scaled version to check the status of
- h- the height of the scaled version to check the status of
- o- the ImageObserver object to be notified as the image is being prepared

Returns:

the bitwise inclusive OR of the ImageObserver ([II-§2.11](#)) flags for the data that is currently available.

See Also:

prepareImage ([II-§3.6.17](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.createImage

public abstract Image createImage(ImageProducer producer)

Creates an image from the specified image producer.

Parameters:

producer- the image producer

Returns:

the image produced

public abstract Image createImage(int width, int height)

Creates an off-screen drawable image to be used for double buffering.

Parameters:

width- the specified width

height- the specified height

Returns:

an off-screen drawable image.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.disable

public abstract void disable()

Makes the component insensitive to user input.

See Also:

enable [\(II-§3.6.6\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.dispose

public abstract void dispose()

Disposes of the component peer and any resources used by it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.enable

public abstract void enable()

Makes the component sensitive to user input. This is the default.

See Also:

disable [\(II-§3.6.4\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.getColorModel

public abstract ColorModel getColorModel()

Gets the component's color model ([II-§2.1](#)), which is an abstract class that encapsulates how to translate between pixel values of an image and its red, green, blue, and alpha components.

Returns:

the color model of the component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.getFontMetrics

public abstract FontMetrics getFontMetrics(Font font)

Parameters:

font- the font

Returns:

the font metrics for this component; if the component is not currently on the screen, this method returns null.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.getGraphics

public abstract Graphics getGraphics()

Returns:

the graphics context of this component; this method returns null if the component is not currently on the screen.

See Also:

paint (II-§3.6.15).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.getToolkit

public abstract Toolkit getToolkit()

Returns:

the toolkit that component peer is part of.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.handleEvent

public abstract boolean handleEvent(Event e)

This method is called when any event occurs inside the component.

The method must return true to indicate that it has successfully handled the action; or false if the event that triggered the action should be passed up to the component's containing object.

Parameters:

e- the event

Returns:

false if the event is to be given to the component's containing object; true if the event has been handled and no further action is necessary.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.hide

public abstract void hide()

Hides the component.

See Also:

show [\(II-§3.6.26\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.minimumSize

public abstract Dimension minimumSize()

Returns:

the minimum size of this component.

See Also:

preferredSize [\(II-§3.6.16\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.nextFocus

public **abstract void nextFocus()**

Moves the focus to the next component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.paint

public abstract void paint(Graphics g)

Paints the component.

The (0, 0) coordinate of the graphics context is the top-left corner of the components. The clipping region of the graphics context is the rectangle of the component.

Parameters:

g— the graphics context to use for painting.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.preferredSize

public abstract Dimension preferredSize()

Returns:

the preferred size of this component.

See Also:

minimumSize [\(II-§3.6.13\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.prepareImage

public abstract boolean

prepareImage(Image img, int w, int h, ImageObserver o)

Prepares an image for rendering on this component at the specified width and height.

The image data is downloaded asynchronously in another thread and an appropriately scaled screen representation of the image is generated.

Parameters:

image- the image to prepare a screen representation for

width- the width of the desired screen representation

height- the height of the desired screen representation

observer- the ImageObserver object to be notified as the image is being prepared

Returns:

true if the image has already been fully prepared; false otherwise.

See Also:

ImageObserver (II-§2.11).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.print

public abstract void print(Graphics g)

Prints this component.

The (0, 0) coordinate of the graphics context is the top-left corner of the components. The clipping region of the graphics context is the rectangle of the component.

See Also:

paint (II-§3.6.15).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.repaint

```
public abstract void repaint(long tm, int x, int y, int width, int height)
```

Repaints the specified rectangle of the component.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the component

height- the height of the component

See Also:

paint [\(II-§3.6.15\)](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ComponentPeer.requestFocus

public abstract void requestFocus()

Requests the input focus.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.reshape

public abstract void

reshape(int x, int y, int width, int height)

Reshapes the component to the specified bounding rectangle.

Parameters:

x- the x coordinate

y- the y coordinate

width- the width of the component

height- the height of the component

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.setBackground

public abstract void setBackground(Color c)

Sets the background color for the component.

Parameters:

c – the color

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.setFont

public abstract void setFont(Font f)

Sets the font of the component.

Parameters:

f – the font

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.setForeground

public abstract void setForeground(Color c)

Sets the foreground color for the component.

Parameters:

c – the color

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ComponentPeer.show

public abstract void show()

Shows the component; if the component had been made invisible by a call to the hide method (II-§3.6.12), makes the component visible again.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.7 Interface ContainerPeer

```
public interface java.awt.peer.ContainerPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract Insets insets(); §3.7.1
}
```

The container peer interface specifies the methods that all implementations of Abstract Window Toolkit containers must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ContainerPeer.insets

public **abstract Insets insets()**

Determines the insets of the container, which indicate the size of the border of the container.

A frame, for example, has a top inset that corresponds to the height of the frame's title bar.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.8 Interface DialogPeer

```
public interface java.awt.peer.DialogPeer
    extends java.awt.peer.WindowPeer (II-§3.22)
{
    // Methods
    public abstract void setResizable(boolean resizable); §3.8.1
    public abstract void setTitle(String title); §3.8.2
}
```

The dialog window peer interface specifies the methods that all implementations of Abstract Window Toolkit dialog windows must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DialogPeer.setResizable

public abstract void setResizable(boolean resizable)

Sets the resizable flag.

Parameters:

resizable- true if the dialog window is resizable; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

DialogPeer.setTitle

public abstract void setTitle(String title)

Sets the title of the dialog window.

Parameters:

title- the new title

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.9 Interface FileDialogPeer

```
public interface java.awt.peer.FileDialogPeer
    extends java.awt.peer.DialogPeer (II-§3.8)
{
    // Methods
    public abstract void setDirectory(String dir); §3.9.1
    public abstract void setFile(String file); §3.9.2
    public abstract void §3.9.3
        setFilenameFilter(FilenameFilter filter);
}
```

The file dialog window peer interface specifies the methods that all implementations of Abstract Window Toolkit file dialog windows must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FileDialogPeer.setDirectory

public abstract void setDirectory(String dir)

Sets the directory of the file dialog window to be the specified directory.

Parameters:

dir- the specific directory

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialogPeer.setFile

public abstract void setFile(String file)

Sets the selected file for the file dialog window to be the specified file.

Parameters:

file- the file being set

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FileDialogPeer.setFilenameFilter

public abstract void

setFilenameFilter(FilenameFilter filter)

Sets the filename filter ([I-§2.26](#)) for this file dialog window to the specified filter.

Parameters:

filter- the specified filter

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.10 Interface FramePeer

```
public interface java.awt.peer.FramePeer
    extends java.awt.peer.WindowPeer (II-§3.22)
{
    // Methods
    public abstract void setCursor(int cursorType); §3.10.1
    public abstract void setIconImage(Image im); §3.10.2
    public abstract void setMenuBar(MenuBar mb); §3.10.3
    public abstract void setResizable(boolean resizeable); §3.10.4
    public abstract void setTitle(String title); §3.10.5
}
```

The frame peer interface specifies the methods that all implementations of Abstract Window Toolkit frames must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


FramePeer.setCursor

public abstract void setCursor(int cursorType)

Sets the cursor image to be one of the predefined cursors.

Parameters:

cursorType- one of the predefined cursor constants

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FramePeer.setIconImage

public abstract void setIconImage(Image im)

Sets the image to display when this frame is iconized.

Note that not all platforms support the concept of iconizing a window.

Parameters:

image- the icon image to be displayed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FramePeer.setMenuBar

public abstract void setMenuBar(MenuBar mb)

Sets the menu bar of this frame to the specified menu bar.

Parameters:

mb – the new menu bar

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FramePeer.setResizable

public abstract void setResizable(boolean resizable)

Determines whether this frame should be resizable.

Parameters:

`resizable`- true if the frame should be resizable; false otherwise

{ewl msdncd.dll, ewcright, /c"Microsoft"}

FramePeer.setTitle

public abstract void setTitle(String title)

Sets the title of this frame to the specified title

Parameters:

title- the new title of this frame, or null to delete the title.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.11 Interface LabelPeer

```
public interface java.awt.peer.LabelPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void setAlignment(int alignment); §3.11.1
    public abstract void setText(String label); §3.11.2
}
```

The label peer interface specifies the methods that all implementations of Abstract Window Toolkit label must define.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LabelPeer.setAlignment

public abstract void setAlignment(int alignment)

Sets the alignment for this label to the specified alignment.

Parameters:

alignment- the alignment value

{ewl msdncd.dll, ewcright, /c"Microsoft"}

LabelPeer.setText

public abstract void setText(String label)

Sets the text for this label to the specified text.

Parameters:

label- the text that makes up the label

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.12 Interface ListPeer

```
public interface java.awt.peer.ListPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void addItem(String item, int index); §3.12.1
    public abstract void clear(); §3.12.2
    public abstract void delItems(int start, int end) §3.12.3
    public abstract void deselect(int index); §3.12.4
    public abstract int[] getSelectedIndexes(); §3.12.5
    public abstract void makeVisible(int index); §3.12.6
    public abstract Dimension minimumSize(int v); §3.12.7
    public abstract Dimension preferredSize(int v); §3.12.8
    public abstract void select(int index); §3.12.9
    public abstract void setMultipleSelections(boolean v); §3.12.10
}
```

The scrolling list peer interface specifies the methods that all implementations of Abstract Window Toolkit scrolling lists must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ListPeer.addItem

public abstract void addItem(String item, int index)

Adds the specified string to the scrolling list at the specified position.

The index argument is 0-based. If the index is -1, or greater than or equal to the number of items already in the list, then the item is added at the end of the list.

Parameters:

item- the string to be added

index- the position at which to put in the item

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.clear

public abstract void clear()

Removes all items from the scrolling list.

See Also:

delItems (II-§3.12.3).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.delItems

public abstract void delItems(int start, int end)

Deletes the items in the range start is less than or equal to item is less than or equal to item from the scrolling list.

Parameters:

start- the index of the first element to delete

end- the index of the last element to delete

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.deselect

public abstract void deselect(int index)

Deselects the item at the specified index.

Parameters:

index- the position of the item to deselect

See Also:

select [\(II-§3.12.9\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.getSelectedIndexes

public abstract int[] getSelectedIndexes()

Returns:

an array of the selected indexes on the scrolling list.

See Also:

select (II-§3.12.9)

deselect (II-§3.12.4).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.makeVisible

public abstract void makeVisible(int index)

Forces the item at the specified index to be visible.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.minimumSize

public abstract Dimension minimumSize(int v)

Parameters:

`rows` - number of rows

Returns:

the minimum dimensions needed to display the specified number of rows in a scrolling list.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.preferredSize

public abstract Dimension preferredSize(int v)

Parameters:

`rows` - number of rows

Returns:

the preferred dimensions needed to display the specified number of rows in a scrolling list.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.select

public abstract void select(int index)

Selects the item at the specified index.

Parameters:

index- the position of the item to select

See Also:

deselect ([II-§3.12.4](#)).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ListPeer.setMultipleSelections

public abstract void setMultipleSelections(boolean v)

Sets whether this scrolling list allows multiple selections.

Parameters:

v – if true then multiple selections are allowed; otherwise, only one item can be selected at a time

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.13 Interface MenuBarPeer

```
public interface java.awt.peer.MenuBarPeer
    extends java.awt.peer.MenuComponentPeer (II-§3.14)
{
    // Methods
    public abstract void addHelpMenu(Menu m); §3.13.1
    public abstract void addMenu(Menu m); §3.13.2
    public abstract void delMenu(int index); §3.13.3
}
```

The menu bar peer interface specifies the methods that all implementations of Abstract Window Toolkit menu bars must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuBarPeer.addHelpMenu

public abstract void addHelpMenu(Menu m)

Sets the help menu on the menu bar to be the specified menu.

Parameters:

m- the help menu

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuBarPeer.addMenu

public abstract void addMenu(Menu m)

Adds the specified menu to the menu bar.

Parameters:

m – the menu to be added

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuBarPeer.delMenu

public abstract void delMenu(int index)

Removes the menu located at the specified index from the menu bar.

Parameters:

index- the position of the menu to be removed

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.14 Interface MenuComponentPeer

```
public interface java.awt.peer.MenuComponentPeer
{
    // Methods
    public abstract void dispose(); §3.14.1
}
```

The menu component peer interface specifies the methods that all implementations of Abstract Window Toolkit menu components must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuComponentPeer.dispose

public abstract void dispose()

Disposes of the menu component peer and any resources used by it.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.15 Interface MenuItemPeer

```
public interface java.awt.peer.MenuItemPeer
    extends java.awt.peer.MenuComponentPeer (II-§3.14)
{
    // Methods
    public abstract void disable(); §3.15.1
    public abstract void enable(); §3.15.2
    public abstract void setLabel(String label); §3.15.3
}
```

The menu item peer interface specifies the methods that all implementations of Abstract Window Toolkit menu items must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuItemPeer.disable

public abstract void disable()

Disables this menu item. It can no longer be selected by the user.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItemPeer.enable

public abstract void enable()

Enables this menu item. It can be selected by the user.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

MenuItemPeer.setLabel

public abstract void setLabel(String label)

Changes the menu item's label to be the specified string.

Parameters:

label- the new label, or null for no label

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.16 Interface MenuPeer

```
public interface java.awt.peer.MenuPeer
    extends java.awt.peer.MenuItemPeer (II-§3.15)
{
    // Methods
    public abstract void addItem(MenuItem item); §3.16.1
    public abstract void addSeparator(); §3.16.2
    public abstract void delItem(int index); §3.16.3
}
```

The menu peer interface specifies the methods that all implementations of Abstract Window Toolkit menus must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuPeer.addItem

public abstract void addItem(MenuItem item)

Adds the specified menu item to this menu.

Parameters:

item- the menu item to be added

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuPeer.addSeparator

public abstract void addSeparator()

Adds a separator line to this menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


MenuPeer.delItem

public abstract void delItem(int index)

Deletes the item at the specified index from this menu.

Parameters:

index- an index in the menu

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.17 Interface PanelPeer

```
public interface java.awt.peer.PanelPeer  
    extends java.awt.peer.ContainerPeer (II-§3.7)  
{  
}
```

The panel peer interface specifies the methods that all implementations of Abstract Window Toolkit panels must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.18 Interface ScrollbarPeer

```
public interface java.awt.peer.ScrollbarPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract void setLineIncrement(int l); §3.18.1
    public abstract void setPageIncrement(int l); §3.18.2
    public abstract void setValue(int value); §3.18.3
    public abstract void setValues(int value, int visible §3.18.4
                                int minimum, int maximum);
}
```

The scroll bar peer interface specifies the methods that all implementations of Abstract Window Toolkit scroll bars must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ScrollbarPeer.setLineIncrement

public abstract void setLineIncrement(int l)

Sets the line increment of the scroll bar.

The line increment is the value that is added to or subtracted from the value of the scroll bar when the user clicks the line down or line up gadget.

Parameters:

l- the new line increment

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ScrollbarPeer.setPageIncrement

public abstract void setPageIncrement(int l)

Sets the page increment of the scroll bar.

The page increment is the value that is added to or subtracted from the value of the scroll bar when the user clicks the page down or page up gadget.

Parameters:

l – the new page increment

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ScrollbarPeer.setValue

public abstract void setValue(int value)

Sets the value of the scroll bar to the specified value.

Parameters:

value- the new value of the scroll bar

{ewl msdncd.dll, ewcright, /c"Microsoft"}

ScrollbarPeer.setValues

```
public abstract void setValues(int value, int visible, int minimum, int maximum)
```

Sets several parameters of the scroll bar simultaneously.

Parameters:

value- the value of the scroll bar

visible- the amount visible per page

minimum- the minimum value of the scroll bar

maximum- the maximum value of the scroll bar

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.19 Interface TextAreaPeer

```
public interface java.awt.peer.TextAreaPeer
    extends java.awt.peer.TextComponentPeer (II-§3.20)
{
    // Methods
    public abstract void insertText(String txt, int pos); §3.19.1
        public abstract Dimension §3.19.2
            minimumSize(int rows, int cols);
    public abstract Dimension §3.19.3
        preferredSize(int rows, int cols);
    public abstract void replaceText(String txt, int start, §3.19.4
        int end);
}
```

The text area peer interface specifies the methods that all implementations of Abstract Window Toolkit text areas must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextAreaPeer.insertText

public abstract void insertText(String txt, int pos)

Inserts the specified text at the specified position.

Parameters:

str- the text to insert

pos- the position at which to insert the text

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextAreaPeer.minimumSize

public abstract Dimension minimumSize(int rows, int cols)

Returns:

the minimum dimensions needed for the text area.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextAreaPeer.preferredSize

public abstract Dimension preferredSize(int rows, int cols)

Returns:

the preferred dimensions needed for the text area.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextAreaPeer.replaceText

public abstract void

replaceText(String txt, int start, int end)

Replaces the text from the start (inclusive) index to the end (exclusive) index with the new text specified.

Parameters:

str- the replacement text

start- the start position

end- the end position

See Also:

insertText [\(II-§3.19.1\)](#).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.20 Interface TextComponentPeer

```
public interface java.awt.peer.TextComponentPeer
    extends java.awt.peer.ComponentPeer (II-§3.6)
{
    // Methods
    public abstract int getSelectionEnd(); §3.20.1
    public abstract int getSelectionStart(); §3.20.2
    public abstract String getText(); §3.20.3
    public abstract void select(int selStart, int selEnd); §3.20.4
    public abstract void setEditable(boolean editable) §3.20.5
    public abstract void setText(String t); §3.20.6
}
```

The text component peer interface specifies the methods that all implementations of Abstract Window Toolkit text components must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextComponentPeer.getSelectionEnd

public abstract int getSelectionEnd()

Returns:

selected text's end position.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponentPeer.getSelectionStart

public abstract int getSelectionStart()

Returns:

selected text's start position.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponentPeer.getText

public abstract String getText()

Returns:

the text of this text component.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponentPeer.select

public abstract void select(int selStart, int selEnd)

Selects the text in the text component from the start (inclusive) index to the end (exclusive) index.

Parameters:

selStart- the start position of the text to select

selEnd- the end position of the text to select

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponentPeer.setEditable

public abstract void setEditable(boolean editable)

Sets the text component to be user editable if the boolean argument is true. If the flag is false, sets the text component so that the user cannot change its contents.

Parameters:

editable- a flag indicating whether the text component should become user editable

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextComponentPeer.setText

public abstract void setText(String 1)

Sets the text of this text component to the specified text.

Parameters:

1- the new text

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


§3.21 Interface TextFieldPeer

```
public interface java.awt.peer.TextFieldPeer
    extends java.awt.peer.TextComponentPeer (II-§3.20)
{
    // Methods
    public abstract Dimension minimumSize(int cols); §3.21.1
    public abstract Dimension preferredSize(int cols); §3.21.2
    public abstract void setEchoCharacter(char c); §3.21.3
}
```

The text field peer interface specifies the methods that all implementations of Abstract Window Toolkit text fields must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TextFieldPeer.minimumSize

public abstract Dimension minimumSize(int cols)

Parameters:

cols- the number of columns

Returns:

the minimum dimensions needed to display the text field with the specified number of columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextFieldPeer.preferredSize

public abstract Dimension preferredSize(int cols)

Parameters:

cols- the number of columns

Returns:

the preferred dimensions needed to display the text field with the specified number of columns.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

TextFieldPeer.setEchoCharacter

public abstract void setEchoCharacter(char c)

Sets the echo character for this text field. Any character that the user types in the text field is echoed in the text field as the echo character.

An echo character is useful for fields where the user input shouldn't be echoed to the screen such as in the case of a text field for typing in a password.

Parameters:

c – the echo character for this text field

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§3.22 Interface WindowPeer

```
public interface java.awt.peer.WindowPeer
    extends java.awt.peer.ContainerPeer (II-§3.7)
{
    // Methods
    public abstract void toBack(); §3.22.1
    public abstract void toFront(); §3.22.2
}
```

The window peer interface specifies the methods that all implementations of Abstract Window Toolkit windows must define.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


WindowPeer.toBack

public abstract void toBack()

Sends this window to the back.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

WindowPeer.toFront

public abstract void toFront()

Brings this window to the front.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Package java.applet

Classes

4.1 Class Applet

Interfaces

4.2 Interface AppletContext

4.3 Interface AppletStub

4.4 Interface AudioClip

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.1 Class Applet

```
public class java.applet.Applet
    extends java.awt.Panel (II-§1.33)
{
    // Constructors
    public Applet(); §4.1.1

    // Methods
    public void destroy(); §4.1.2
    public AppletContext getAppletContext(); §4.1.3
    public String getAppletInfo(); §4.1.4
    public AudioClip getAudioClip(URL url); §4.1.5
    public AudioClip getAudioClip(URL url, String name); §4.1.6
    public URL getCodeBase(); §4.1.7
    public URL getDocumentBase(); §4.1.8
    public Image getImage(URL url); §4.1.9
    public Image getImage(URL url, String name); §4.1.10
    public String getParameter(String name); §4.1.11
    public String[][] getParameterInfo(); §4.1.12
    public void init(); §4.1.13
    public boolean isActive(); §4.1.14
    public void play(URL url); §4.1.15
    public void play(URL url, String name); §4.1.16
    public void resize(Dimension d); §4.1.17
    public void resize(int width, int height); §4.1.18
    public final void setStub(AppletStub stub); §4.1.19
    public void showStatus(String msg); §4.1.20
    public void start(); §4.1.21
    public void stop(); §4.1.22
}
```

An applet is a small program that is not intended to be run on its own, but to be embedded inside another application.

The Applet class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer. The Applet class provides a standard interface between applets and their environment.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Applet.Applet

public `Applet()`

The default constructor for an applet.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Applet.destroy

public void destroy()

This method is called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated. The stop (II-§4.1.22)method will always be called before destroy.

A subclass of Applet should override this method if it has any operation that it wants to perform before it is destroyed. For example, an applet with threads would use the init (II-§4.1.13)method to create the threads and the destroy method to kill them.

The implementation of this method provided by the Applet class does nothing.

See Also:

start (II-§4.1.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getAppletContext

public AppletContext getAppletContext()

Determines this applet's context, which allows the applet to query and affect the environment in which it runs.

This environment of an applet represents the document that contains the applet.

Returns:

the applet's context.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getAppletInfo

public String getAppletInfo()

Returns information about this applet. An applet should override this method to return a String containing information about the author, version, and copyright of the applet.

The implementation of this method provided by the Applet class returns null.

Returns:

a string containing information about the author, version and copyright of the applet.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Applet.getAudioClip

public AudioClip **getAudioClip**(URL url) :

Returns the AudioClip object (II-§3.4) specified by the URL argument.

This method always returns immediately, whether or not the audio clip exists. When this applet attempts to play the audio clip, the data will be loaded.

Parameters:

url- an absolute URL giving the location of the audio clip

Returns:

the audio clip at the specified URL.

public AudioClip **getAudioClip**(URL url, String name)

Returns the AudioClip object (II-§3.4) specified by the URL and name arguments.

This method always returns immediately, whether or not the audio clip exists. When this applet attempts to play the audio clip, the data will be loaded.

Parameters:

url- an absolute URL giving the base location of the audio clip

name- the location of the audio clip, relative to the url argument

Returns:

the audio clip at the specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getCodeBase

public URL getCodeBase ()

Returns:

the URL (I-§4.8) of this applet.

See Also:

getDocumentBase (II-§4.1.8).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getDocumentBase

public URL `getDocumentBase()`

Returns:

the URL (I-§4.8) of the document that contains this applet.

See Also:

`getCodeBase` (II-§4.1.7).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getImage

public Image getImage(URL url)

Returns an Image object (II-§1.24) that can then be painted on the screen. The url that is passed as an argument must specify an absolute URL.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url- an absolute URL giving the location of the image

Returns:

the image at the specified URL.

public Image getImage(URL url, String name)

Returns an Image object (II-§1.24) that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url- an absolute URL giving the base location of the image

name- the location of the image, relative to the url argument

Returns:

the image at the specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getParameter

public String **getParameter**(String name)

Returns the value of the named parameter in the HTML tag. For example, if this applet is specified as:

```
<applet code="Clock"  >  
<param name=Color value="blue">  
</applet>
```

A call to `getParameter("Color")` returns the value "blue".

The name argument is case insensitive.

Parameters:

name- a parameter name

Returns:

the value of the named parameter.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.getParameterInfo

public String[][] getParameterInfo()

Returns information about the parameters that can be understood by this applet. An applet should override this method to return an array of arrays of Strings describing these parameters.

Each element of the array should be a set of three Strings containing the name, the type, and a description. For example:

```
String pinfo[][] = {
    {"fps",          "1-10",          "frames per second"},
    {"repeat",       "boolean",       "repeat image loop"},
    {"imgs",         "url",           "images directory"}
};
```

The implementation of this method provided by the Applet class returns null.

Returns:

an array describing the parameters this applet looks for.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.init

public void init()

This method is called by the browser or applet viewer to inform this applet that it has been loaded into the system. It is always called before the first time the start method (II-§4.1.21) is called.

A subclass of Applet should override this method if it has initialization to perform. For example, an applet with threads would use the init method to create the threads and the destroy (II-§4.1.2) method to kill them.

The implementation of this method provided by the Applet class does nothing.

See Also:

stop (II-§4.1.22).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.isActive

public boolean isActive()

Determines if this applet is active. An applet is marked active just before its start method (II-§4.1.21) is called. It becomes inactive immediately after its stop (II-§4.1.22) method is called.

Returns:

true if the applet is active; false otherwise.

See Also:

start (II-§4.1.21).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.play

public void play(URL url)

Plays the audio clip at the specified absolute URL. Nothing happens if the audio clip cannot be found.

Parameters:

url- an absolute URL giving the location of the audio clip

public void play(URL url, String name)

Plays the audio clip given the URL and a specifier that's relative to it. Nothing happens if the audio clip cannot be found.

Parameters:

url- an absolute URL giving the base location of the audio clip

name- the location of the audio clip, relative to the url argument

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.resize

public void **resize**(Dimension d)

Requests that this applet be resized.

Parameters:

d- An object giving the new width and height

Overrides:

resize in class Component (II-§1.10.65).

public void **resize**(int width, int height)

Requests that this applet be resized.

Parameters:

width- the new requested width for the applet

height- the new requested height for the applet

Overrides:

resize in class Component (II-§1.10.66).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.setStub

public final void setStub(AppletStub stub)

Sets this applet's stub. This is done automatically by the system.

Parameters:

stub- The new stub

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Applet.showStatus

public void showStatus(String msg)

Requests that the argument string be displayed in the "status window." Many browsers and applet viewers provide such a "status window" where the application can inform users of its current state.

Parameters:

msg- a string to display in the status window

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.start

public void start()

This method is called by the browser or applet viewer to inform this applet that it should start its execution. It is called after the init method and each time the applet is revisited in a Web page.

A subclass of Applet should override this method if it has any operation that it wants to perform each time the Web page containing it is visited. For example, an applet with animation might want to use the start method to resume animation, and the stop method (II-§4.1.22) to suspend the animation.

The implementation of this method provided by the Applet class does nothing.

See Also:

init (II-§4.1.13)
destroy (II-§4.1.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Applet.stop

public void stop()

This method is called by the browser or applet viewer to inform this applet that it should stop its execution. It is called when the Web page that contains this applet has been replaced by another page and also just before the applet is to be destroyed.

A subclass of Applet should override this method if it has any operation that it wants to perform each time the Web page containing it is no longer visible. For example, an applet with animation might want to use the start (II-§4.1.21) method to resume animation, and the stop method to suspend the animation.

The implementation of this method provided by the Applet class does nothing.

See Also:

init (II-§4.1.13)
destroy (II-§4.1.2).

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.2 Interface AppletContext

```
public interface java.applet.AppletContext
{
    // Methods
    public abstract Applet getApplet(String name); §4.2.1
    public abstract Enumeration getApplets(); §4.2.2
    public abstract AudioClip getAudioClip(URL url); §4.2.3
    public abstract Image getImage(URL url); §4.2.4
    public abstract void showDocument(URL url); §4.2.5
    public abstract void §4.2.6
        showDocument(URL url, String target);
    public abstract void showStatus(String status); §4.2.7
}
```

This interface corresponds to an applet's environment: the document containing the applet and the other applets in the same document.

The methods in this interface can be used by an applet to obtain information about its environment.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


AppletContext.getApplet

public abstract Applet getApplet(String name)

Finds and returns the applet in the document represented by this applet context with the given name. The name can be set in the HTML tag by setting the name attribute.

Parameters:

name- an applet name

Returns:

the applet with the given name, or null if not found.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


AppletContext.getApplets

public abstract Enumeration getApplets()

Finds all the applets in the document represented by this applet context.

Returns:

an enumeration of all applets in the document represented by this applet context.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletContext.getAudioClip

public abstract AudioClip getAudioClip(URL url)

Creates an audio clip.

Parameters:

`url` - an absolute URL giving the location of the audio clip

Returns:

the audio clip at the specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletContext.getImage

public **abstract** **Image** **getImage**(URL url)

Returns an Image object (II-§1.24) that can then be painted on the screen. The url argument that is passed as an argument must specify an absolute URL.

This method always returns immediately, whether or not the image exists. When the applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url- an absolute URL giving the location of the image.

Returns:

the image at the specified URL.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletContext.showDocument

public **abstract void** **showDocument**(URL url)

Replaces the Web page currently being viewed with the given URL. This method may be ignored by applet contexts that are not browsers.

Parameters:

url- an absolute URL giving the location of the document

public **abstract void** **showDocument**(URL url, String target)

Requests that the browser or applet viewer show the Web page indicated by the url argument. The target argument indicates where to display the frame. The target argument is interpreted as follows:

"_self"	show in the current frame
"_parent"	show in the parent frame
"_top"	show in the top-most frame
"_blank"	show in a new unnamed top-level window
name	show in a new top-level window named <i>name</i>

An applet viewer or browser is free to ignore.

Parameters:

url- an absolute URL giving the location of the document

target- a String indicating where to display the page

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletContext.showStatus

public abstract void showStatus(String status)

Requests that the argument string be displayed in the "status window." Many browsers and applet viewers provide such a "status window" where the application can inform users of its current state.

Parameters:

status- a string to display in the status window

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.3 Interface AppletStub

```
public interface java.applet.AppletStub
{
    // Methods
    public abstract void appletResize(int width, int height); §4.3.1
    public abstract AppletContext getAppletContext(); §4.3.2
    public abstract URL getCodeBase(); §4.3.3
    public abstract URL getDocumentBase(); §4.3.4
    public abstract String getParameter(String name); §4.3.5
    public abstract boolean isActive(); §4.3.6
}
```

When an applet is first created, an applet stub is attached to it using the applet's setStub method (II-§4.1.19). This stub serves as the interface between the applet and the browser environment or applet viewer environment in which the application is running.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


AppletStub.appletResize

public abstract void appletResize(int width, int height)

Called when the applet wants to be resized.

Parameters:

width- the new requested width for the applet

height- the new requested height for the applet

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletStub.getAppletContext

public abstract AppletContext getAppletContext()

Returns:

the applet's context.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletStub.getCodeBase

public abstract URL getCodeBase()

Returns:

the URL of the applet.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletStub.getDocumentBase

public abstract URL getDocumentBase()

Returns:

the URL of the document containing the applet.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletStub.getParameter

public **abstract** **String** **getParameter**(**String** name)

Returns the value of the named parameter in the HTML tag. For example, if an applet is specified as:

```
<applet code="Clock" value="blue">  
</applet>
```

A call to `getParameter("Color")` returns the value "blue".

Parameters:

name- a parameter name

Returns:

the value of the named parameter.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AppletStub.isActive

public abstract boolean isActive()

Determines if the applet is active. An applet is active just before its start method (II-§4.1.21) is called. It becomes inactive immediately after its stop (II-§4.1.22) method is called.

Returns:

true if the applet is active; false otherwise.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

§4.4 Interface AudioClip

```
public interface java.applet.AudioClip
{
    // Methods
    public abstract void loop();      §4.4.1
    public abstract void play();     §4.4.2
    public abstract void stop();     §4.4.3
}
```

The AudioClip interface is a simple abstraction for playing a sound clip. Multiple AudioClip items can be playing at the same time, and the resulting sound is mixed together to produce a composite.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


AudioClip.loop

public abstract void loop()

Starts playing this audio clip in a loop.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AudioClip.play

public abstract void play()

Starts playing this audio clip. Each time this method is called, the clip is restarted from the beginning.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

AudioClip.stop

```
public abstract void stop()
```

Stops playing this audio clip.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


An Introduction to Visual J++

Microsoft Visual J++ is an integrated Windows-hosted development tool for Java programming. It allows you to create a new Java program, make modifications and compile, run, and debug it, all within a single environment.

Visual J++ is built around Developer Studio, Microsoft's common development environment. If you've previously used other Microsoft products built around Developer Studio—such as Visual C++—you already know how to find your way around the Visual J++ environment. If you've never used a Developer Studio-based product, this test drive will provide an introduction to using Visual J++.

This test drive describes four scenarios:

- Building a sample Java applet
- Creating a new applet with Applet Wizard
- Creating dialogs and menus with the Resource Wizard
- Making simple changes to a project

For more detailed information on using the environment, see the *Visual J++ User's Guide*.

For information about the Java language, see the printed book *Learn Java Now*, or the online versions of Sun Microsystems's *Java Language Specification* and [Java Application Programming Interface](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Building a Sample Java Applet

To get a feel for the basic functionality that Visual J++ offers, you will build one of the sample applets. For the purposes of this example, you will use the WireFrame sample applet supplied with Sun's Java Development Kit. (If you have another applet you'd like to use instead of WireFrame, simply make the appropriate replacements in the remaining procedures.)

This scenario has the following steps:

- Extracting the sample
- Creating the project
- Adding the sample files to the project
- Building the applet
- Specifying the HTML page
- Running the applet

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Extracting the Sample

The first step in building the sample applet is to install the WireFrame sample onto your hard disk. All the sample programs are available through the online documentation system.

{ewl msdncd, EWGraphic, tut2a 0 /a "build.bmp"} To extract the WireFrame sample

- 1** Click on the InfoView tab of the Project Workspace window.
- 2** Open the Samples node.
- 3** Open the Sun Samples node.
- 4** Open the WireFrame node.
A window appears, displaying an abstract (or description) of the WireFrame sample applet.
- 5** Click on the button in the sample abstract.
- 6** Select all the files except the .MAK and .MDP files. (To perform multiple selection with the mouse, hold down the mouse button while selecting. To perform multiple selection with the keyboard, hold down the SHIFT key while using the cursor keys.)
You will recreate the .MAK and .MDP files in the step Creating a Java Project.
- 7** Choose Copy to copy the files.
By default, the files get copied to the directory \MSDev\Samples\Sun\WireFrame. You can specify a different directory if you want.
- 8** Choose OK.
The sample files are copied to your hard disk.
- 9** Choose Close to dismiss the Sample Application dialog.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a Java Project

The Visual J++ development environment uses projects to keep track of the files needed to build a Java program and store information about the classes.

{ewl msdncd, EWGraphic, tut3a 0 /a "build.bmp"} To create a project for WireFrame

- 1 From the File menu, choose New.

The New dialog box appears.

- 2 In the New box, select Project Workspace.
- 3 Choose OK.

The New Project Workspace dialog box appears.

- 4 In the Name text box, type "WireFrame".

- 5 From the Type list, select Java Workspace.

If you're using Visual J++ by itself, without having another product like Visual C++ installed, the Type list displays only Java-specific types of projects. If you have other products installed as well, you'll see many other types of projects listed.

In the Platforms list box, the Java Virtual Machine check box is checked; this refers to the interpreter that Java programs run on.

- 6 The Location text box should read "\MSDev\Samples\Sun\WireFrame" (or whatever directory you installed the sample files in). If you want to change the directory, use the Browse button.
- 7 Choose Create.

Visual J++ creates a new project named "WireFrame."

If you look in the WireFrame subdirectory now, you'll see several new files, including WireFrame.mak, WireFrame.ncb, and others. Visual J++ uses these files to manage information about a project.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding Files to the Project

The next step is to add the WireFrame files to the project you've just created.

{ewl msdncd, EWGraphic, tut4a 0 /a "build.bmp"} To add the files to the WireFrame project

- 1 From the Insert menu, choose Files into Project.
The Insert Files into Project dialog box appears.
- 2 In the File Name field, type "*.java;*.html".
The file list displays the .JAVA and *.HTML files in the directory.
- 3 Select all the files. (To perform multiple selection with the mouse, hold down the mouse button while selecting the files. To perform multiple selection with the keyboard, hold down the SHIFT key while using the cursor keys.)
- 4 Choose OK.
Visual J++ adds the files to the project.

You can now browse your project using the various panes of the Project Workspace window. For example, when you click on the FileView tab, you'll see a node labeled WireFrame files. Double-click on the node to expand it, and you'll see a node for each file in the project. Double-click on a filename, and a source window appears, displaying the contents of the file. Notice that in .JAVA files, all keywords are in blue and ordinary comments are in green; the special "doc comments" in Java are in gray. This is the default syntax coloring that Visual J++ provides for Java source files. Visual J++ also provides syntax coloring for HTML files, which you can see by opening an .HTML file.

If you click on the ClassView tab of the Project Workspace window, you'll see a node labeled WireFrame classes. Double-click on the node to expand it, and you'll see nodes labeled FileFormatException, Matrix3D, Model3D, and ThreeD. These are the classes defined by the WireFrame applet. If you double-click on a node, you'll see a list of all the methods and variables defined by that class. If you double-click on any of those, the window displaying the source for that class will display the declaration of that method or variable.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Building the Applet

Once the files have been added to the project, you're ready to compile the .JAVA files into .CLASS files.

{ewl msdncd, EWGraphic, tut5a 0 /a "build.bmp"} To build the applet

- From the Build menu, choose Build WireFrame.
This command builds the entire project. You can also compile individual source files using the Compile command from the Build menu.

Note With the Java language, there isn't a one-to-one mapping between .JAVA files and .CLASS files. The Java compiler creates a .CLASS file for each class declaration in the .JAVA file. In our example, there are only two .JAVA files, but within them are four class declarations. As a result, the compiler produces four .CLASS files. Similarly, the names of the .CLASS files generated by the compiler reflect the names of the classes declared in the source code, not necessarily the filenames of the .JAVA files.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Specifying the HTML Page

An applet is normally executed in the context of an HTML page, viewed in a browser. Before you can run your applet within a browser, you have to specify the HTML page to be used. By default, Visual J++ will look for an HTML page with the same name as the **Applet**-derived class: in this case, "threed.html." However, you may wish to specify a different HTML page. The following procedure explains how to do that:

{ewl msdncd, EWGraphic, tut6a 0 /a "build.bmp"} To specify the applet's HTML page

- 1 From the Build menu, choose Settings.

The Project Settings dialog box appears.

- 2 Choose the Debug tab.
- 3 In the drop-down list labeled Category, choose Browser.
- 4 Select the button "Use parameters from HTML page."
- 5 In the edit box labeled HTML Page, type Example1.HTML. This file is included with the WireFrame applet.

If you don't have an HTML page but you want to run your Java program as an applet, Visual J++ will create a temporary HTML page for you. You can specify the parameters for the temporary page by selecting the button "Enter parameters below" and using the grid control beneath it.

You can also set other options from this dialog box, such as the browser to use for viewing the HTML page.

- 6 Choose OK.

You're now ready to run the applet.

Note that the WireFrame sample comes with several HTML files, each providing different input to the WireFrame applet.

Visual J++ also lets you run an applet without an HTML page, using a tool called JView. This tool is described in Running the Application.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Running the Applet

Running an applet is typically done by using a Web browser to load an HTML page that references the applet. Visual J++ uses Internet Explorer as the default browser for running Java applets.

{ewl msdncd, EWGraphic, tut7a 0 /a "build.bmp"} To run the applet

- 1 From the Build menu, choose Execute WireFrame.

The information for the Running Class dialog box appears.

- 2 Type threed (which is the name of the **Applet**-derived class in the WireFrame sample) and click OK.

Visual J++ now launches Internet Explorer to open the HTML page Example1.HTML, which loads and runs the WireFrame applet. The applet displays a wireframe cube, which you can rotate by clicking and dragging the mouse in the window.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating an Applet with Applet Wizard

For creating new Java projects, Visual J++ provides Applet Wizard, a tool that generates a set of starter files. You can use these files as a basis for developing your own Java applet.

The Applet Wizard offers several options for the applet that it generates:

- The ability to run as an application as well as an applet (see Applets and Applications and More About Applets and Applications).
- Explanatory comments in the source code.
- A sample HTML page for running the applet.
- Basic multithreading support.
- A sample animation.
- Handlers for mouse events.
- Code for reading parameters from the HTML page.
- Code for offering author and copyright information.

Using these options can reduce the amount of generic code you need to type when developing your own applet. More information about these options is given on the Applet Wizard dialogs themselves, and in the comments for the source code.

Before proceeding, you should know how to build a sample Java applet. This scenario has the following steps:

- Running Applet Wizard
- Running an Application

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Applets and Applications

There are two basic types of Java programs: applets and applications. Applets can be referenced by an HTML page and executed when that page is loaded by a Java-capable browser, such as Internet Explorer. Applications do not require a browser to run; they can be executed from the command line using a stand-alone interpreter.

In terms of source code, the basic differences between an applet or an application are as follows:

- An applet must define a class derived from the **Applet** class. An application can do the same, but it is not required to do so.
- An application must define a class with a method named **main** that controls its execution. An applet does not use a **main method**; its execution is controlled by various methods defined by the **Applet** class.

Visual J++'s Applet Wizard can create either an ordinary Java applet, or an applet that can also be run as an application.

Since the previous scenario demonstrated running an ordinary applet, this scenario will demonstrate running an applet as an application.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Running Applet Wizard

{ewl msdncd, EWGraphic, tut10a 0 /a "build.bmp"} To create an Applet Wizard project

- 1** From the File menu, choose New.
The New dialog box appears.
- 2** In the New box, select Project Workspace.
- 3** Choose OK.
The New Project Workspace dialog box appears.
- 4** In the Name text box, type "NewApp".
- 5** From the Type list, select Java Applet Wizard.
The Platforms box displays a check box for Java Virtual Machine; this refers to the interpreter that Java programs run on.
- 6** The Location text box should read "\MSDev\Projects\NewApp"; if you want your project in a different directory, type in the path you want, or use the Browse button to choose a directory.
- 7** Choose Create.
Visual J++ creates a workspace and the first Java Applet Wizard dialog box appears.
- 8** Where the dialog says, "How would you like to be able to run your program?", choose "As an applet and as an application."
When you choose this option, the Applet Wizard creates an application that can run either as an applet, embedded in an HTML page, or as a standalone application.
- 9** Choose Finish.
The New Project Information dialog box appears, summarizing the settings and features Applet Wizard will generate for you when it creates your project. If you want different options, you can cancel Applet Wizard and run it again.
- 10** Choose OK.
Applet Wizard creates all the necessary files and opens the project.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Running the Application

You can build an Applet Wizard-created application using the same command you used for building the sample applet. From the Build menu, choose Build. You're now ready to run the application.

{ewl msdncd, EWGraphic, tut11a 0 /a "build.bmp"} To run the application

- 1 From the Build menu, choose Settings.
The Project Settings dialog box appears.
- 2 Choose the Debug tab.
- 3 In the drop-down list labeled Category, the choice displayed should be General. If it is not, choose General.
- 4 In the group box "Debug project under," select the radio button "Stand-alone interpreter."

Note If you try to run a class as an application when the class doesn't have a **main** function defined, the operation will fail. A Java application, by definition, must define a **main** function.

- 5 Choose OK.
- 6 From the Build Menu, choose Execute NewApp.
Visual J++ now runs the application. The application opens a window and displays the string "Loading images..." momentarily. Then it displays the sample animation, a spinning globe.

The Visual J++ Java interpreter is named JView. You can run an application from the command line by typing "JView" and the name of the class that defines a **main** function. (Note that you must set the CLASSPATH environment variable before running JView from the command line. This isn't necessary when you run an application from within the Visual J++ environment, because Visual J++ uses a class path stored separately.)

The program you've created with Applet Wizard can be run both as an applet and as an application. You can demonstrate this by going to the Project Settings dialog again, choosing the Browser button instead, and then running NewApp again.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

More About Applets and Applications

The program created by Applet Wizard can run both as an applet and as an application because it contains an **Applet**-derived class that defines a **main method**. The fact that the class is derived from **Applet** allows it to be executed in the context of an HTML page; in such a situation, its **main method** is simply ignored. The fact that it contains a **main method** allows it to be executed as a standalone application.

If you try running the program as an applet, you'll notice that it looks essentially the same as when it's run as an application. This is noteworthy because, by default, there's another fundamental difference between applets and applications: an applet automatically has a graphical user interface (a portion of the HTML page that contains it), while an application is text based by default.

Applet Wizard has generated code to handle this difference, giving the program a graphical user interface even when it's run as an application. Applet Wizard does this by defining a class derived from the **Frame** class. This class, in conjunction with the program's **Applet**-derived class, provides a window for the application, allowing the same graphical user interface code to run. This frame window class is used only when the program is executed standalone.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Wizard

With traditional Windows development tools, user-interface elements like menus and dialogs are specified using resource files that are separate from the source files. In Java, by contrast, menus and dialogs are created in the source code, using Java's Abstract Window Toolkit (AWT). Resource Wizard is a tool for converting Windows resource templates into equivalent Java code.

Before proceeding, you should know how to create a new applet with Applet Wizard. For the purposes of this scenario, create a new project called Demo and specify the following options when running Applet Wizard:

- In Step 1, where the dialog says, "How would you like to be able to run your program?", choose "As an applet only."
- In Step 3, where the dialog says, "Would you like support for animation?", choose "No, thank you."

Once you've created this project, you are ready to proceed with using Resource Wizard.

This scenario has the following steps:

- Creating a resource template
- Running the Resource Wizard
- Using the Resource Wizard-generated dialog
- Using the Resource Wizard-generated menu
- Viewing the results

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Creating a Resource Template

To create the resource template that Resource Wizard will use as input, use the Visual J++ resource editors to create a resource template (.RCT) file.

{ewl msdncd, EWGraphic, tut14a 0 /a "build.bmp"} To create a resource template

- 1** From the File menu, choose New.
The New dialog box appears.
- 2** Choose Resource Template and then choose OK.
Visual J++ opens a resource template window.
- 3** From the Insert menu, choose Resource.
The Insert Resource dialog box appears.
- 4** Choose Dialog and then choose OK.
Visual J++ opens a dialog editing window.
- 5** Press ALT+ENTER to display the property page for the dialog. Change the ID of the dialog to NewDialog.
- 6** Add an edit box, a static text control, and a push button to the dialog using the control palette.
While the dialog editor allows you to add any type of control to the dialog, only controls that have analogs in Java's Abstract Window Toolkit (AWT) will be converted by Visual J++'s Resource Wizard. For a list of supported controls, see Using Java GUI Components. For information on using the Visual J++ dialog editor, see Using the Dialog Editor.
- 7** Close the dialog editing window.
- 8** From the Insert menu, choose Resource.
The Insert Resource dialog box appears.
- 9** Choose Menu and then choose OK.
Visual J++ opens a menu editing window.
- 10** Press ALT+ENTER to display the property page for the menu. Change the ID of the menu to NewMenu.
- 11** Add two new menus, each containing two menu items. Give the menus and menu items whatever names you want.
For information on using the Visual J++ menu editor, see Using the Menu Editor.
- 12** Close the menu editing window.
- 13** From the File menu, choose Save.
The File Save dialog box appears.
- 14** Change the directory to the "c:\MSDev\Projects\Demo", or whatever directory your project resides in.
The default filename that Visual J++ suggests is "Templ1.RCT"; you can change this to whatever name you choose.
- 15** Choose OK.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Running Resource Wizard

The next step is to run Resource Wizard, which reads the resource template file and generates Java code that produces the same user interface.

{ewl msdncd, EWGraphic, tut15a 0 /a "build.bmp"} To run Resource Wizard

- 1 From the Tools menu, choose Java Resource Wizard.
The Java Resource Wizard appears.
- 2 In the File name edit box, specify the .RCT file you created previously, or choose the Browse button to find it.
- 3 Choose Finish.
A dialog box appears, displaying the names of the files generated.
- 4 Choose OK.

For each dialog or menu in the resource template, Resource Wizard creates a .JAVA file containing a class whose name is the ID of the dialog or menu. In addition, when there are dialogs in the resource template, Resource Wizard also creates a separate file for a `DialogLayout` class, which is used by the dialog class(es) to manage the layout of controls.

There's no need to edit the source code in the Resource Wizard-generated .JAVA file(s). If you want to make any changes to your menu or dialog, use the menu editor or dialog editor, re-save the resource template, and run Resource Wizard to regenerate the .JAVA file(s).

Note Each time you run Resource Wizard, it overwrites the previous version of the .JAVA file(s) it generates. As a result, any changes you make to the generated .JAVA file(s) will be lost the next time you run Resource Wizard.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

What Resource Wizard Generates

Dialog Templates

In the Java Abstract Window Toolkit (AWT), user-interface controls—such as buttons and scrollbars—can be arranged within various containers, such as panels, applets, windows, and dialogs. Because dialogs are not the only kind of container, Resource Wizard does not generate a class derived from the AWT's **Dialog** class. Instead, Resource Wizard generates a class that adds controls to an arbitrary container to make it resemble the dialog template. In this manner, you can use Visual J++'s dialog editor to design the appearance of not just dialogs, but also applets, panels, or windows.

This “control creator” class generated by Resource Wizard has two methods: a constructor that takes a **Container object** as an argument, and a `CreateControls` method. Resource Wizard also generates a class named `DialogLayout`, which implements the AWT **LayoutManager** interface. This class is a custom layout manager which allows the position of controls to be specified in the same way that Windows resource files do. Any “control creator” classes generated by Resource Wizard use this layout manager.

Menu Templates

In the Java AWT, menus must appear within a frame, described by the AWT's **Frame** class. Resource Wizard generates a class that adds menus to an arbitrary frame.

This “menu creator” class generated by Resource Wizard has two methods: a constructor that takes a **Frame object** as an argument, and a `CreateMenu` method.

For more information on the Java AWT, see the printed book *Learn Java Now* and the section on the AWT in Sun's Java Application Programming Interface.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Wizard-generated Dialog

The first step is to add the Resource Wizard-generated file(s) to your project using the same procedure you performed to add the sample files to a project. If you have been following the steps described in this scenario, the files you must add are named NewDialog.JAVA, DialogLayout.JAVA, and NewMenu.JAVA.

The next step is to add code that actually references the classes created by Resource Wizard.

{ewl msdncd, EWGraphic, tut19a 0 /a "build.bmp"} To use the dialog class in your Java program

- 1 Open the Java source file Demo.JAVA and add the following **import** statement to the beginning of the file:

```
import NewDialog;
```

- 2 Near the beginning of the applet declaration, after the declaration of the **Thread variable**, add the following line:

```
NewDialog dlg;
```

- 3 In the applet's init method, remove the call to **resize** and add the following lines:

```
dlg = new NewDialog( this );  
dlg.CreateControls();
```

The first line allocates an instance of the `NewDialog` class, passing the **this pointer** to the constructor, specifying the applet as the container. The second line calls the `CreateControls` method, which adds all the controls to the applet.

- 4 Comment out the call to **drawString** in the paint method.
- 5 Just before the end of the applet declaration, add the following code:

```
public boolean action( Event evt, Object arg )  
{  
    if ( evt.target instanceof Button )  
    {  
        String val = dlg.IDC_EDIT1.getText();  
        dlg.IDC_STATIC1.setText( val );  
        return true;  
    }  
    return false;  
}
```

The **action method** responds to events generated by controls. In this example, the method checks if the event's **target** was a **Button object**. (Since the dialog in this example has only one button, it's not necessary to test which button was clicked.) If so, the method reads the value of the **TextField object** and uses it to set the value of the **Label object**. The names of the **TextField** and **Label** objects correspond to the IDs of the edit control and static text control, respectively, in the original dialog resource.

- 6 From the File menu, choose Save.


```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Using the Resource Wizard-generated Menu

{ewl msdncd, EWGraphic, tut20a 0 /a "build.bmp"} To use the menu class in your Java program

- 1 In the Java source file Demo.JAVA, add the following **import** statement to the beginning of the file:

```
import NewMenu;
```

- 2 In the applet declaration, after the declaration of the `NewDialog` variable, add the following line:

```
NewMenu menu;  
MyFrame frame;
```

These lines declare variables for using the `NewMenu` class.

- 3 In the applet's init method, after the lines that create the dialog, add the following lines:

```
frame = new MyFrame( "menu test" );  
frame.resize( 100, 100 );  
menu = new NewMenu( frame );  
menu.CreateMenu();  
frame.show();
```

The `MyFrame` class will be defined in the next step. These lines declare an instance of `MyFrame`, specify it as the frame for an instance of the `NewMenu` class, initialize the menus, and display the frame window.

- 4 At the end of the file, add the following code:

```
class MyFrame extends Frame  
{  
    String text = new String(" ");  
  
    MyFrame(String title)  
    {  
        super(title);  
    }  
  
    public void paint(Graphics g)  
    {  
        g.drawString(text, 10, 10);  
    }  
  
    public boolean action( Event evt, Object obj )  
    {  
        Object target = evt.target;  
        if ( target instanceof MenuItem)  
        {  
            text = (String)obj;  
            repaint();  
            return true;  
        }  
        return false;  
    }  
}
```



```
    }  
}
```

The `MyFrame` class extends the **Frame** class and declares a **String variable**. The override of the **paint method** displays the value of this string. The override of the **action method** responds to events generated within the frame. The **method** checks if the event's **target** was a **MenuItem object**. If so, the **method** sets the `text` **variable** to the **label** of the menu item that was chosen, which is passed in the `obj` parameter.

- 5 From the File menu, choose Save.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Viewing the Results

Build your project using the same command you used for [building the sample applet](#). From the Build menu, choose Build.

You are now ready to run the application. Choose Execute Demo from the Build menu. First Internet Explorer appears, displaying an edit control, a static text field, and a push button on the HTML page; these are the controls specified in the dialog resource. A separate window also appears, displaying a menu bar; these are the menus specified in the menu resource.

To test the [event handler](#) for the dialog, type “hello” in the edit control and click on the push button. This causes the contents of the static text field to change to “hello,” reflecting the value in the edit control. You can easily add more controls and then expand the [action method](#) to examine which control was chosen and perform different actions accordingly.

To test the [event handler](#) for the menus, choose a command from one of the menus in the frame window. This causes the name of the last menu item chosen to appear in the window. You can easily expand the [action method](#) to examine which menu item was chosen and perform different actions accordingly.

Note The resource [template](#) (.RCT) file you created is not part of the Java project; only the .JAVA files generated from the the resource [template](#) are part of the project. Consequently, if you make modifications to the resource template—for example, by editing a menu or dialog box—the resource [template](#) does not automatically get converted the next time you [build](#) your project. You must manually run Resource [Wizard](#) to regenerate the .JAVA files from the resource [template](#); those .JAVA files will be automatically recompiled the next time you [build](#) the project.

For more information on the Resource [Wizard](#), see [Using the Resource Wizard](#).

For more information on the Java AWT, see the printed book *Learn Java Now* and the section on the AWT in Sun's [Java Application Programming Interface](#).

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Making Changes to a Java Project

Visual J++ provides features that simplify the creation of certain elements of a Java program:

- Adding a new class
- Adding a new method
- Adding a new variable

Visual J++ also includes a bitmap editor for creating images for use with your Java program.

- Creating a new image

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Adding a Class

{ewl msdncd, EWGraphic, tut23a 0 /a "build.bmp"} To add a class

- 1** From the Insert menu, choose New Class. (Or, in the ClassView pane, point the cursor at the name of a class, click the right mouse button, and choose Create New Class).
- 2** The Create New Class dialog appears.
- 3** Fill in the dialog:
 - Enter the name of the class in the Name edit box.
 - If the new class extends an existing one, enter the class's superclass in the Extends edit box.
 - Enter the package you want the class to be part of, if it's not the default one.
 - Specify any modifiers (public, **abstract**, or **final**) for the class by selecting the appropriate check box.
- 4** Choose OK.

Visual J++ creates a new source file containing the declaration of the class you've specified. This file is automatically added to the project.

For information about Java classes and Java packages, see the printed book *Learn Java Now* or the online *Java Language Specification*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding a Method

{ewl msdncd, EWGraphic, tut24a 0 /a "build.bmp"} To add a method

- 1** In the ClassView pane, point the cursor at the name of the class you want to add a method to and click the right mouse button.
- 2** In the pop-up menu, choose Add Method.
The Add Method dialog appears.
- 3** Fill in the dialog
 - In the Return Type box, enter the return type of the method.
 - In the Method Declaration box, enter the name of the method. If you want the method to have parameters, specify them enclosed in parentheses.
 - In the Access area, choose the degree of access you want the method to have.
 - In the Modifier area, choose any modifiers you want to apply to the method.
- 4** Choose OK.

Visual J++ adds an empty definition for the method you've specified at the top of the class declaration.

For information about Java methods, access specifiers, and modifiers, see the printed book *Learn Java Now* or the online *Java Language Specification*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Adding a Variable

{ewl msdncd, EWGraphic, tut25a 0 /a "build.bmp"} To add a variable

- 1** In the ClassView pane, point the cursor at the name of the class you want to add a variable to and click the right mouse button.
- 2** In the pop-up menu, choose Add Variable.
The Add Variable dialog appears.
- 3** Fill in the dialog
 - In the Variable type box, enter the type of the variable.
 - In the Variable name box, enter the name of the variable.
 - If you want the variable initialized to a particular value, enter it in the Initial value box.
 - From the Access drop-down list, choose the degree of access you want the variable to have.
 - In the Modifier area, choose any modifiers you want to apply to the variable.

- 4** Choose OK.

Visual J++ adds a declaration for the variable you've specified at the top of the class definition.

For information about Java variables, access specifiers, and modifiers, see the printed book *Learn Java Now* or the online *Java Language Specification*.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Creating a New Image

Java currently understands two formats for graphic images: Graphic Interface Format (GIF) or Joint Photographic Experts Group (JPEG) format. Visual J++ supports the creation of images in these formats, as well as common Windows formats like Bitmap (.BMP) and Device-Independent Bitmap (.DIB).

{ewl msdncd, EWGraphic, tut26a 0 /a "build.bmp"} To create a new image

- 1** From the File menu, choose New.
The New dialog box appears.
- 2** In the New box, select Bitmap.
A bitmap editing window appears.
- 3** Use the palettes available to draw your bitmap.
- 4** From the File menu, choose Save.
The Save dialog box appears.
- 5** Type the name you want to save the bitmap under, and give it the extension ".GIF" or ".JPG" depending on whether you want to save it as a .GIF or a .JPEG file.
- 6** Choose OK.

To add a .GIF or .JPG file to your project, pull down the Insert menu and choose Files into Project.

For more information on creating images, see Using the Graphic Editor.

{ewl msdncd.dll, ewcright, /c"Microsoft"}

Animator: Demonstrates Animation

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the Animator project files." /I"1001" /Bcodeview}
```

Animator is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `animator` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ArcTest: Demonstrates Arc Drawing

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the ArcTest project files." /I"1002" /Bcodeview}
```

ArcTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `arctest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BarChart: Demonstrates Chart Drawing

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the BarChart project files." /I"1003" /Bcodeview}
```

BarChart is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `chart` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Blink: Demonstrates Blinking Text

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the Blink project files." /I"1004" /Bcodeview}
```

Blink is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `blink` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


BouncingHeads: Demonstrates Animation

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the BouncingHeads project files." /I"1005"  
/Bcodeview}
```

BouncingHeads is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `bounceitem` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CardTest: Demonstrates CardLayout

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the CardTest project files." /I"1006" /Bcodeview}
```

CardTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `cardtest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DitherTest: Demonstrates Color Dithering

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the DitherTest project files." /I"1007"  
/Bcodeview}
```

DitherTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `dithertest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DrawTest: A Line-Drawing Program

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the DrawTest project files." /I"1008" /Bcodeview}
```

DrawTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `drawtest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Fractal: A Fractal-Drawing Program

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the Fractal project files." /I"1009" /Bcodeview}
```

Fractal is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `clsfractal` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GraphicsTest: Demonstrates Graphics Primitives

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the GraphicsTest project files." /I"1010"  
/Bcodeview}
```

GraphicsTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `graphicstest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


GridLayout: Demonstrates Auto-Layout Algorithm

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the GridLayout project files." /I"1011"  
/Bcodeview}
```

GridLayout is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `graph` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageMap: Demonstrates Images

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the ImageMap project files." /I"1012"  
/Bcodeview}
```

ImageMap is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `imagemap` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ImageTest: Demonstrates Image Manipulation

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the ImageTest project files." /I"1013"  
/Bcodeview}
```

ImageTest is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `imagetest` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


JumpingBox: Demonstrates Mouse Tracking

```
{ewc msdn.cd.dll, ewbutton, /T"Click to open or copy the JumpingBox project files." /I"1014"  
/Bcodeview}
```

JumpingBox is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `mousetrack` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdn.cd.dll, ewcright, /c"Microsoft"}
```


MoleculeViewer: Demonstrates 3D Modeling

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the MoleculeViewer project files." /I"1015"  
/Bcodeview}
```

MoleculeViewer is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends java.applet.Applet). Enter xyzapp as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


NervousText: Demonstrates Text Animation

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the NervousText project files." /I"1016"  
/Bcodeview}
```

NervousText is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `nervoustext` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


ScrollingImages: Demonstrates Scrolling

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the ScrollingImages project files." /I"1017"  
/Bcodeview}
```

ScrollingImages is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `imagetape` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SimpleGraph: Demonstrates Graphing

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the SimpleGraph project files." /I"1018"  
/Bcodeview}
```

SimpleGraph is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `graphapplet` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


SpreadSheet: A Simple Spreadsheet Program

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the SpreadSheet project files." /I"1019"  
/Bcodeview}
```

SpreadSheet is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `spreadsheet` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TicTacToe: A Simple TicTacToe Program

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the TicTacToe project files." /I"1020"  
/Bcodeview}
```

TicTacToe is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `tictactoe` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


TumblingDuke: Demonstrates Animation

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the TumblingDuke project files." /I"1021"  
/Bcodeview}
```

TumblingDuke is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends Java.applet.Applet). Enter tumbleitem as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


UnderConstruction: Demonstrates Animation and Sound

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the UnderConstruction project files." /I"1022"  
/Bcodeview}
```

UnderConstruction is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `jackhammerduke` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


WireFrame: Demonstrates 3D Modeling

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the WireFrame project files." /I"1023"  
/Bcodeview}
```

WireFrame is a sample from the Java Development Kit from Sun Microsystems.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `threed` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


JavaBeep: Java calling a COM Object

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the JavaBeep project files." /I"1029"  
/Bcodeview}
```

This sample is a simple demonstration of a Java application calling a COM object. The project for this sample registers the COM component and creates a Java description of the COM component. Typically a programmer will perform one or both of these steps explicitly. See the `JavaCallingCOM` sample for an example of performing these steps.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `javabeep` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

Note: Due to security restrictions, you can run sample applets that use COM services only from within Developer Studio. In general, applets that use COM services must reside on the class path, or originate in a .CAB file having a digital signature.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


JavaCallingCOM: Java calling a COM Object

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the JavaCallingCOM project files." /I"1030"  
/Bcodeview}
```

This sample is a simple demonstration of a Java application calling a COM object.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


Scripting: VBScript driving a Java Applet

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the Scripting project files." /I"1032" /Bcodeview}
```

This sample consists of a simple Java applet and an HTML page that uses VBScript to drive the applet.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. Choose the Open Workspace... item from the File menu.
8. In the Open Project Workspace dialog box, navigate to the directory for the sample project.
9. Select the sample's .MDP file and click Open.
10. Choose the Execute item from the Build menu.
11. Click Yes when asked to rebuild the out of date files.
12. A dialog asks for the name of the class file to run (you must specify the class that extends `java.applet.Applet`). Enter `displaytext` as the class-file name and click OK.

That's it! You should now see the applet running inside your default browser.

To debug the applet:

1. Exit the browser.
2. Choose the Debug/Step Into item from the Build menu.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


OLEControls: Java Manipulating an OLE Control

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the OLEControls project files." /I"1026"  
/Bcodeview}
```

Demonstrates the use of the outline control (implemented by MSOUTL32.OCX) by a Java applet.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


HTMLControls: Demonstrates Use of INTRINSC.OCX

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the HTMLControls project files." /I"1028"  
/Bcodeview}
```

Demonstrates the use of HTML controls (implemented by INTRINSC.OCX) by a Java applet.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


DAOSample: Demonstrates DAO

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the DAOSample project files." /I"1027"  
/Bcodeview}
```

Demonstrates the use of Data Access Objects (DAO) from Java.

This sample requires that you have DAO installed on your machine. You can install DAO from Visual J++'s Setup program by choosing Custom Install and checking the Database Options checkbox.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


RDOSample: Demonstrates RDO

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the RDOSample project files." /I"1031"  
/Bcodeview}
```

Demonstrates the use of Remote Data Objects (RDO) from Java.

This sample requires that you have RDO installed on your machine. You can install RDO from Visual J++'s Setup program by choosing Custom Install and checking the Database Options checkbox.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


COMCallingJava: Implementing a COM Object in Java

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the COMCallingJava project files." /I"1025"  
/Bcodeview}
```

This sample is a simple demonstration of using Java to implement a COM object that can be called from a Visual Basic client application.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```


CabAndSign: Creating a Signed CAB File

```
{ewc msdncd.dll, ewbutton, /T"Click to open or copy the CabAndSign project files." /I"1024"  
/Bcodeview}
```

This sample is a simple demonstration of packaging a Java applet into a cabinet (.CAB) file and digitally signing it.

To run this sample:

1. Print out these instructions.
2. Put the product CD in the drive.
3. Click on the button above.
4. In the Sample Application dialog box, click the Copy All... button.
5. Specify the target directory for the sample project (or accept the default) and click OK.
6. Click Close to close the Sample Application dialog box.
7. In Windows Explorer, go to the directory in which you installed the sample and double-click on README.HTML for further instructions.

```
{ewl msdncd.dll, ewcright, /c"Microsoft"}
```