Forté for Java Community Edition 1.0

Java Integrated Development Environment

User's Guide

Documentation version 0.9.4

Contents

Preface

This book describes how to develop with and customize Forte for Java Community Edition 1.0.

Audience

This document is intended for software engineers who will implement Java™ applications and applets with Forte for Java Community Edition 1.0. It is assumed that the reader has a general knowledge of Java. Java beginners can use this guide in conjunction with documentation for the Java™ 2 Platform.

Conventions Used In This Document

The following conventions are used in the text:

- **This font is used to denote**
    - — items you can select in the GUI, such as buttons and menu choices
- `This mono-spaced font is used to denote`
    - o Examples of Java code
    - o File names

  - Explorer nodes

- Meta-names (e.g. words like *YourName*), which describe the type of text to be entered rather than the literal string, are *italicized*.

"Press ENTER" means that you should press the Enter or Return key on your keyboard. Keys like F5 and F9 refer to those function keys (also sometimes labeled PF5 and PF9). "CTRL" refers to the Control key. A set of keystrokes joined with "+", like CTRL+F9, means you should press the first key (here, CTRL), hold it down, and press the second key (here, F9).

In listed source or command line code, lines indented from the previous ones should be entered on the same line when typed into the Editor window or console. For example, the following should all be typed as one line:

```
C:\jdk1.2.2\jre\bin\java -jar "C:\Program
     Files\Forte4J\modules\openfile.jar" -port 2121 "C:\My
     Development\com\mycom\Foo.java"
```

Look and feel

Depending on your platform, the appearance of your Graphical User Interface (GUI) may appear quite different from the screenshots shown here. By default the IDE launches in the Windows look and feel when running on a Windows platform, or the Metal look and feel on all other platforms. Most screenshots in this document were taken using the Metal look and feel (though most shots of the Main Window were made using the Windows look and feel. The look and feel can be set via the **View | Look & Feel** menu.

Note: The Windows look and feel is not available on non-Windows platforms.

Contact Information

For the latest news and information, please check the NetBeans website http://www.netbeans.com/.

If you have any general queries about Forte for Java products and release schedules or would just like to pass along your comments, please mail `feedback@netbeans.com`.

Online Technical Support

If you're having problems with NetBeans, please make use of the following online resources:

Forum

The NetBeans Forum is buzzing with discussion on all aspects of NetBeans Developer, from technical support issues to custom extensions. The web-based format (http://cgi.netbeans.com/cgi-bin/webx/) allows you to search past discussions and issues as well as view conversations in a threaded format. As a registered user, you can customize your interface, mark your messages, and add discussions to the Forum.

If you prefer to follow the newsgroups with a news browser, you can still do that as well. Visit our Newsgroup Information page at http://www.netbeans.com/newsgroups.html to learn more. By using these public forums, you have access to the whole of the international Forte for Java community. The NetBeans Support Team reads and posts to these groups regularly.

The FAQ

Please check the FAQ (Frequently Asked Questions) list to find answers to questions you might have regarding installation, startup, and running Forte for Java Community Edition. This list is updated regularly as new issues come to light. The FAQ can be found at http://www.netbeans.com/faq.html.

Virtual Machines

Check to ensure that you are using a supported JVM (Java Virtual Machine). All known, working JVMs are listed on the JVMs page of the website at http://www.netbeans.com/jvms.html.

Contacting Technical Support

If the FAQ and Forum do not address your questions, you can go to the technical support page of our website at http://www.netbeans.com/support.html. There you will find a host of resources to help you better use Forte for Java Community Edition. You can use the online Support Request form if you wish to directly contact our technical support team. Please fill out all required fields on the form. In particular, it is crucial that you provide your system information and log file. Without this information the Forte for Java Support Team may be unable to handle your request.

Chapter 1

Welcome to Forte™ for Java™ Community Edition 1.0

Forte™ for Java™ Community Edition 1.0 is a Java integrated development environment (IDE) written in Java. Previously known as NetBeans Developer, Forte for Java is a cross-platform tool that is fully functional for client and server side development. Forte for Java takes advantage of the strengths of Java to provide you with a dynamic and responsive environment.

Forte for Java is modular. This means that the IDE functionality for editing, debugging, GUI generation, EJB support, etc. is represented in modules that you can download and update dynamically. Instead of waiting months for a new major release, you can upgrade the latest modules from Sun Microsystems and our partners as soon as they are available.

Forte for Java is extensible in just about every way imaginable. Forte for Java has a complete set of Open APIs that are available to our users and partners – the same set of APIs that our own developers use to build Forte for Java. Partners and individual Forte for Java Community Edition 1.0 for Java users have already built tight integrations with UML tools, debuggers, and native editors. Even complete new Java tools such as visual builders for industrial application domains can take advantage of a mature tool-construction platform!

Forte for Java is customizable. The GUI can be modified to become a reflection of your own development style. You can adjust the look and feel and customize the menus, toolbars, Component Palette, workspaces, and settings.

In addition to this Community Edition, Forte for Java also comes in an Internet Edition. Forte for Java Internet Edition includes all of the features of the Community Edition and adds a host of others, including modules to support development in RMI, CORBA, EJB.

New in Forte for Java Community Edition 1.0

Forte for Java Community Edition incorporates a wide range of new features since the release of NetBeans Developer 2.1 X2, including the following:

Object Browser

With the Object Browser, all of your sources are logically presented in one streamlined view. You can view your applications by package, object, and member (or Bean patterns), as well as apply filters to customize what is displayed. Using context menus in the Object Browser, you can open files, access their property sheets, compile and execute applications, and more.

New Editor features

The Editor has many enhancements, such as:

- Java code completion
- a wider selection of keyboard shortcuts and abbreviations, which you can now easily edit
- line numbering
- bookmarks

In addition, you can set distinct configurations for the editing of Java, HTML, and text files.

Visual GridBag Layout customizer

Now the most complex standard Java layout manager is easier to use with Forte for Java's GridBag layout customizer. The GridBag customizer dialog provides you with a dynamic visual representation of the components in their respective "grid bags" and gives you different ways of adjusting the constraints (such as dragging, direct entry of values, and tool icons.)

Update Center

The Auto Update feature allows you to connect to Forte For Java's website straight from your IDE and download and automatically install new and updated modules.

Source synchronization

The Java source synchronization feature will save you time by automatically generating all of the implementation methods used by your source code.

Javadoc support

With the Javadoc module, you have access to Java API documents from within the API and can automatically have Javadoc comments generated for your files.

JPDA Debugger support

Forte for Java supports Sun's new JPDA debugger, which provides features such as variables on watches and automatic setting of breakpoints on exceptions.

Printing

You can now print your source files and take advantage of a wide array of customizable settings to control the appearance of the printouts.

Other changes since Developer 2.x

Besides the addition of new features, most of the changes since NetBeans Developer 2.x are internal and won't affect the way you work with the IDE. Users familiar with Developer 2.x will have no problem jumping into Forte for Java and benefitting from its improved architecture, which allows it run more smoothly and makes it easier to integrate new modules, whether supplied by Sun or a third-party developer, or created by you. However, users of Developer 2.x should take note of the following modifications.

Explorer organization

The biggest visible change in Forte for Java is the Explorer organization. The Repository node has stayed intact, but the three other main nodes have changed.

- Most of items that were under `Environment` in previous versions now appear in `Global Settings`.
- `Templates`, which used to be one of the main nodes, is now under `Global Settings` as well.
- The settings that previously appeared under `Control Panel` now appear under `Project Settings` along with other settings that can be configured by project.
- The new `Runtime` tab has been added to track all processes and connections, including running applications, debugging, and advanced modules (such as the Jini Browser, RMI Registry, JDBC, etc.).

Enhanced JavaBeans™ Component support

The Bean Wizard in NetBeans Developer 2.x has been replaced by a series of dialogs available from Explorer context menus which give you more options and wider control over the construction of JavaBeans™ Components.

Configuration for Compiling, Running, and Debugging

You can now set the compiler, executor, and debugger (along with their configurations) individually for each file under the **Execution** tab of its property sheet. Under the `Project Settings` node in the Explorer, you can specify which compilers, executors, debuggers are available (under the `Compiler Types`, `Execution Types`, and `Debugger Types` nodes) and general configurations for running (`Execution Settings`) and debugging (`Debugger Settings`).

XML storage of forms

In NetBeans Developer 2.x forms were stored in a binary serialized format, but in Forte for Java they are stored using XML, making them more powerful and robust, as well as readable by humans and simple scripts. When Forte for Java encounters forms that were stored in the previous format, it asks you whether you want to convert the form into the new XML format.

Note: Forms stored in XML are not usable in NetBeans Developer 2.x

Document overview

After this introduction and the following installation chapter, this guide has four main chapters.

- Chapter 3 provides a quick tour of major components and features of the IDE's user interface, and demonstrates practical use of these features. This is a good place to start if you are using the IDE for the first time.

- Chapter 4 is a "how to" chapter, which provides an overview of developing, running, and debugging applications using the Forte for Java IDE.
- Chapter 5 focuses on all aspects of visual development in the IDE.
- Chapter 6 provides a more thorough explanation of the parts and features of the Forte for Java IDE and shows how you can customize it to best suit the way you work.

Chapter 2

Installation

This chapter describes installation of the Forte for Java Community Edition 1.0 IDE. Before installation, please check the system requirements.

System requirements

The installation procedures themselves require a 1.2 Java virtual machine (JVM) installed on your system to run successfully. If you do not yet have a 1.2 JVM, you should install one prior to running the Forte for Java Community Edition 1.0 installation.

NetBeans Developer 2.1 is available for 1.1 JVMs. See the NetBeans website at http://www.netbeans.com/ for more information.

Please check the NetBeans JVMs page (http://www.netbeans.com/jvms.html) for the latest information on recommended VMs for each version as well as links to VM download sites.

Intel 80x86 platforms

Hardware

Minimum configuration: Windows 95/NT: P133 processor, 64MB RAM, 16MB free disk space.

Recommended configuration: Windows 95/NT: P300 processor, 128MB RAM.

Note: If you have 64MB, you should set the batch file startup flag from `-Xmx128m` to -`Xmx64m`.

Software

Forte for Java Community Edition 1.0 requires a Sun-compatible Java Virtual Machine, version 1.2 or later. The Java™ 2 SDK for Windows is available for download from http://java.sun.com/products/jdk/1.2/.

SPARC/Solaris platform

Hardware

Minimum configuration: SparcStation 5: 170 Mhz, 96MB RAM, 16MB free disk space.

Recommended configuration: UltraSPARC 5: 270 Mhz, 128 MB RAM.

Software

Forte for Java Community Edition 1.0 requires the Java™ 2 SDK, v. 1.2 for Solaris. The latest SDK is available for download from http://www.sun.com/solaris/java/.

Notes for Linux users

The Linux JVM is more resource-intensive, so Linux users may want to have a higher memory configuration.

At the time of this writing, a final version of Java 2 is not yet available for Linux. After some configuration, Forte for Java Community Edition 1.0 will run under the current pre-release available from http://www.blackdown.org/, although there are some issues that can limit performance. Please see the NetBeans website for the latest information.

Installation formats

There are several different installation formats of Forte for Java Community Edition 1.0 available. The software in each of these formats is identical; it is merely bundled in different InstallShield formats for ease of use on different platforms. All formats require a 1.2 JVM already installed on your system to install correctly.

Available formats are:

- `forte4j.exe` – a standard InstallShield Win32 self-extracting executable
- `forte4j.sh` – Unix executable
- `forte4j.class` – a Java class installation (usable on all platforms)

For most platforms, there is a choice of installation format. The `forte4j.class` Java installation will run on any platform with a 1.2 JVM installed. If you are unsure which format to use, use this one. A summary of recommended formats is given below:

- Windows 95 / 98 / NT: `forte4j.exe`
- Linux, Solaris, and other Unixes: `forte4j.sh`
- Other: `forte4j.class`

Note: At the time of this writing, no 1.2 JVM is yet available for Macintosh or OS/2. Macintosh or OS/2 users with a 1.1 JVM can currently run Developer 2.1. Check the NetBeans website for more information on Developer 2.1 and the latest news on upcoming JVMs.

Installation procedures for single users

Here are instructions for each installation format listed above.

forte4j.exe

Double-click the `forte4j.exe` file you have saved to your system. This will decompress the InstallShield routine, which will then launch. Follow through the InstallShield wizard dialogs. The install routine will attempt to locate a 1.2 VM on your system; if it fails to locate one, you must browse to locate one before continuing.

Once complete, you will have a shortcut to Forte for Java Community Edition 1.0 on your Desktop. On your Start Menu, you'll also have a Forte for Java entry containing shortcuts to Forte for Java

Community Edition 1.0, the `README` file, and the NetBeans website.

forte4j.sh

Note that the instructions in this section are only for single-user installations. To install for multiple Unix users, see "Multi-user installations" on page 23.

Open a command prompt. Change working directory to the location where you've saved the `forte4j.sh` file. Launch the installation as a non-root user by typing the command:

```
$sh forte4j.sh
```

Before decompressing, `forte4j.sh` will attempt to locate a 1.2 JVM on your system. A menu listing any found, and the option to specify another VM, will be shown. Once one of these is selected, it is used to launch the InstallShield routine. Follow through the InstallShield Wizard dialogs and specify a location under your home directory as the installation directory.

Once complete, use the launch script `netbeans.sh` in your installation directory to launch Forte for Java Community Edition 1.0.

forte4j.class

Open a command prompt, and change working directory to the location you have the `forte4j.class` file saved. If you have a CLASSPATH set, and it does not include the current directory, you will need to add the current directory to your existing CLASSPATH. If you do not have a CLASSPATH set, skip to the next step. To check whether you have a CLASSPATH currently set, type `set` at a command prompt. If you see CLASSPATH listed, you have a CLASSPATH setting.

- To add the current directory to your CLASSPATH on a Windows machine, type:

```
set CLASSPATH=.;%CLASSPATH%
```

- On a Unix machine with a Bourne-type shell (which typically gives a `$` prompt), type:

```
CLASSPATH=.:$CLASSPATH;export CLASSPATH
```

- On a Unix machine with a C-type shell (which typically gives a `%` or `>` prompt), type this command:

```
setenv CLASSPATH .:$CLASSPATH
```

Next, on all platforms, type the following command (and note that you should not include `.class`):

```
java forte4j
```

This assumes you have a 1.2 VM in your path. If you do not, you must specify the full path to the java interpreter executable—for example:

```
    C:\TEMP>C:\jdk1.2\bin\java.exe forte4j
```

This will initiate a standard InstallShield installation routine. Simply follow through the dialogs as normal.

Once complete, use the launch script in your installation directory to launch Forte for Java Community Edition 1.0. If you are installing `forte4j.class` on a Windows machine, you will have shortcuts for launching Forte for Java on your Desktop and under the Start menu.

Multi-user installations

Windows and Unix users may wish to create shared installations, such that multiple users may use the same installation of Forte for Java Community Edition 1.0. Multi-user installations are particularly useful for computer networks at universities.

For more information on multi-user installations, please check the following web page: http://www.netbeans.com/docs/faqs/multiuser.html

Chapter 3

Quick Tour and Tutorial

The Forte for Java Community Edition 1.0 IDE is easy to learn and use. By following the brief tutorial presented in this chapter, you can quickly become familiar with the IDE's main features, and you'll be well on your way to creating your own Java applications.

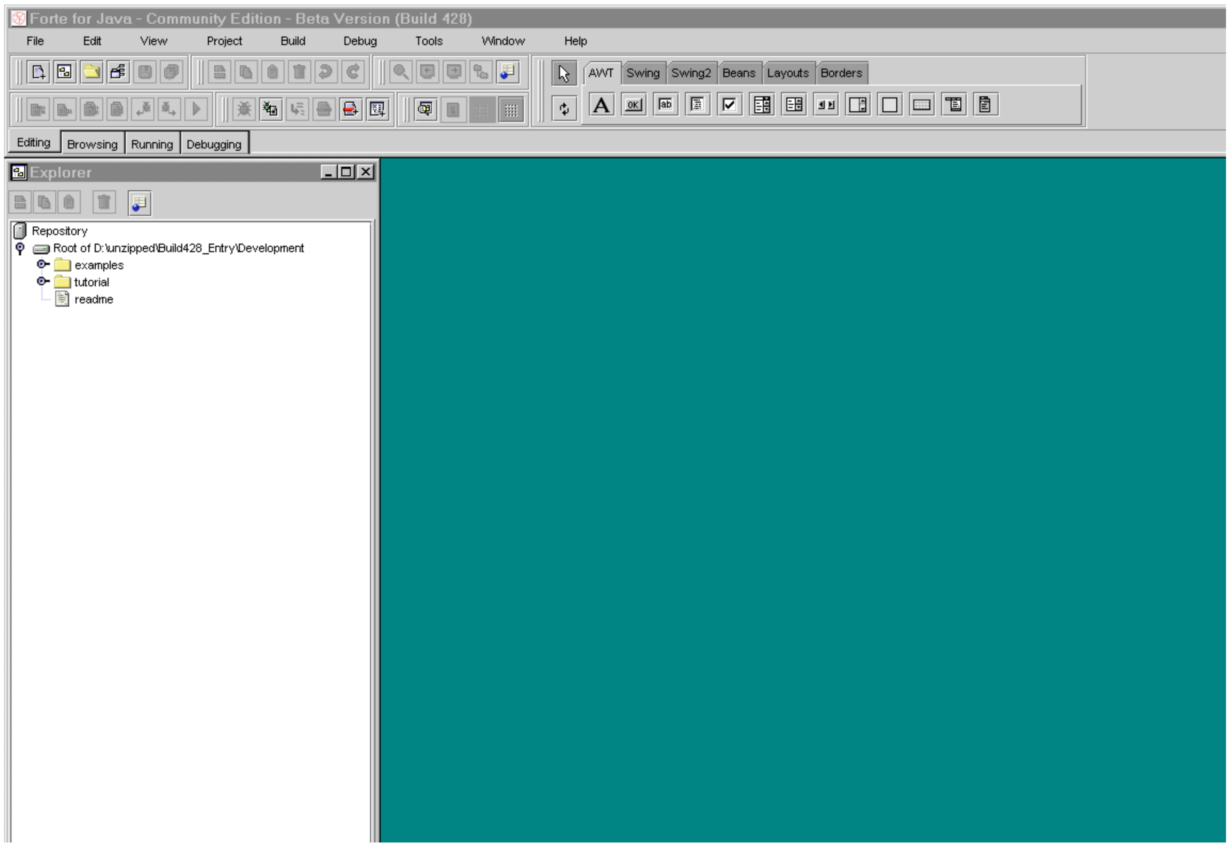
If you don't already have Forte for Java Community Edition 1.0 running, start it now.

Note: This manual is not a comprehensive guide to the Java 2 Platform. For more detailed information about Java, download the documentation that comes with your implementation of the Java 2 Platform. For the Java 2 SDK, Standard Edition, you can download the documentation directly from the Sun website (at press time, http://java.sun.com/products/jdk/1.2/download-docs.html).

The Main Window and Explorer

A few moments after you start Forte for Java Community Edition 1.0, a splash screen will display as the program loads. Then the Main Window will appear at the top of the screen with the Explorer below it, as shown in the figure below.

Forte for Java Community Edition 1.0 at startup



Main Window

The left half of the Main Window is mainly composed of menus and toolbars for controlling the IDE's operation. The right half consists of a special toolbar, the Component Palette, where you can choose objects to add to your visual forms.

Explorer

The Explorer is an object-oriented view of your whole Forte for Java environment containing these four main tabs:

- `Repository`, which stores all files used and created in the IDE
- `Javadoc`, which stores all Javadoc documentation which you create in the IDE.
- `Runtime`, which holds information on all current runtime processes, debugging, and any external services and their connection to the IDE.
- `Project Settings`, which contains nodes for compilation, execution, Editor, printing, and other settings that can be applied to a project (multiple projects can be specified using the

Projects module)

- `Global Settings`, which contains general state and configuration settings for the IDE not specific to projects

Any of these main nodes can be expanded by clicking on the ⊞ sign. Each item found in the Explorer also has its own popup menu which you can access by right-clicking on the node. The icon on the Explorer toolbar is the toggle switch for displaying the Property Sheet pane for the selected node(s).

JavaHelp, context help, and tool tips

As you are getting to know Forte for Java, you may have questions about certain parts of the IDE. Forte for Java's implementation of JavaHelp (with search capability) as well as its context help and tooltips can provide answers to many of your questions.

JavaHelp

You can view the User's Guide through a JavaHelp browser by selecting **Help | Browse Online User's Guide** from the main menu. The JavaHelp viewer is divided into two panes. The left pane lists topics and the right pane provides the content of the topic selected in the left pane.

The toolbar at the top of the left pane has three icons which allow you choose what is shown in the topics list. Clicking the first button displays the User's Guide's table of contents, and clicking the second icon displays the index. The table of contents appears as a tree with expandable nodes and sub-nodes.

To search the documentation, click the third icon and enter your search string in the **Find** field that appears just under the icons. JavaHelp will search the entire User's Guide for that string and then present you with a list of topics pertaining to that string. You can then select one of those topics to have its contents displayed.

Context Help

You can also obtain help for specific features of the IDE by pointing to a specific window, dialog, or icon with the mouse cursor and pressing F1. You can obtain context help for menu items and nodes (such as nodes in the Explorer and Debugger windows) by selecting the node and pressing F1. For menu items, hold the cursor over the menu item so that it is selected, but do not click or release the mouse button to invoke the command.

Tooltips

Tooltips are panels with short text descriptions that appear when you briefly hold the mouse cursor over a part of the IDE. They are particularly useful for explaining the use of tree nodes and individual properties listed in the property sheet.

Creating a program

This section will walk you through the creation of a simple program using the visual features of Forte for Java Community Edition 1.0. This program creates a window with a button that switches the color of a panel when it is pushed.

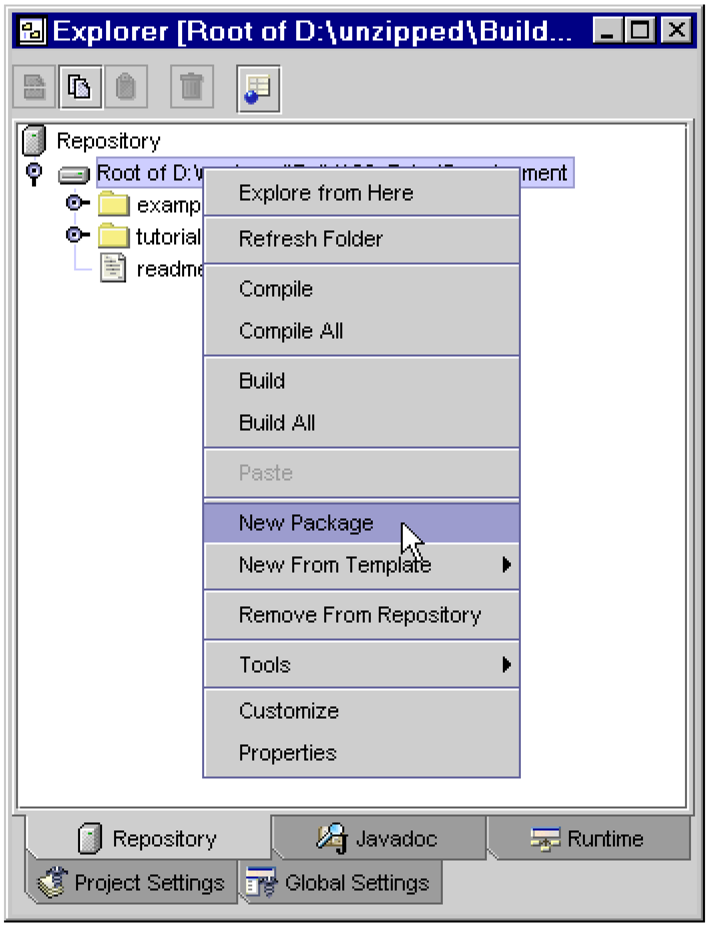
Creating a package

First of all, you'll need to make a place on the file system of your disk to store the files you'll be creating. When you create a Java program, you put it in a **package**. This is a group of one or more related files that make up a working Java program.

We will create a package called `colorswitch`. Under the Repository, you should have a single disk drive icon named for a directory in your Forte for Java installation. We'll put the package there.

To create the package:

1 Right-click on the disk icon and select **New Package** from the popup menu.



2 In the dialogue that appears, type `colorswitch` and click **OK**.

If you expand the disk drive icon, you should see your new package as a folder icon.

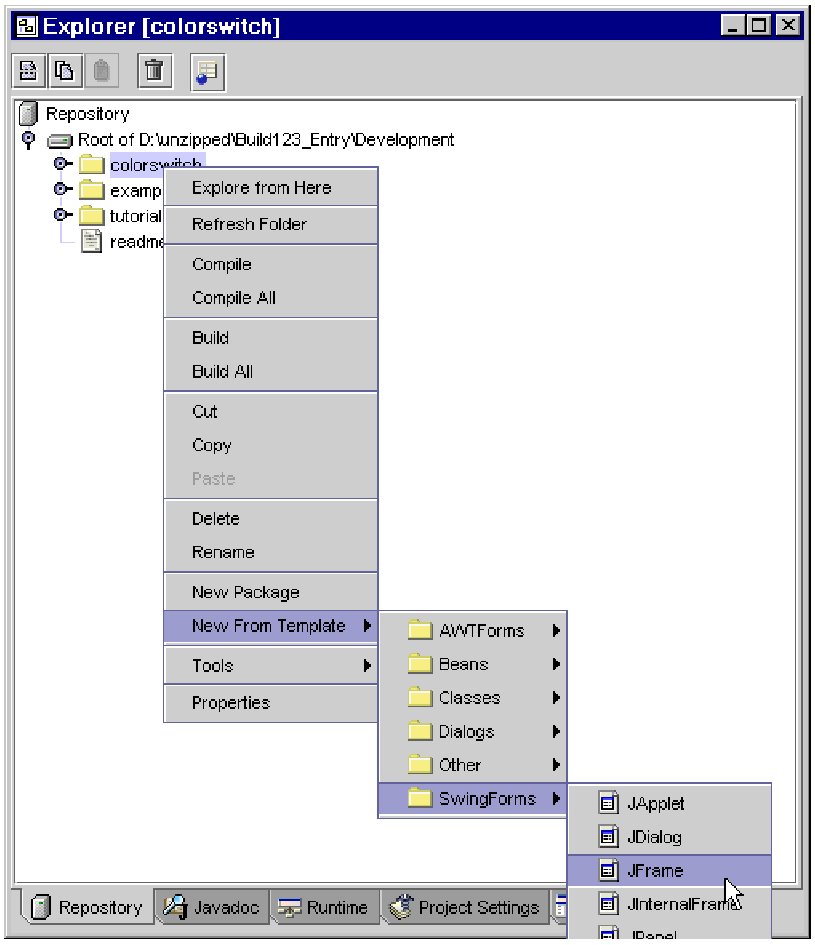
Creating a class

Now we'll create a class to put into the package. We'll use a JFrame, which is one type of **container** (a visual component that can hold other visual components).

To create the JFrame:

1 Right-click on the `colorswitch` package and select **New From Template | SwingForms | JFrame**

from the popup menu.



2 In the dialogue that appears, type `ColorSwitch` for the class name and click **OK**.

The following three windows will then open:

- The **Form Editor window**, which is a visual editor for designing classes.
- The **Component Inspector**, which shows the components of your class in its top pane. In the bottom pane, properties of the component selected in the top pane are displayed.
- The **Editor window** shows the Java source code for the class.

Setting a layout manager

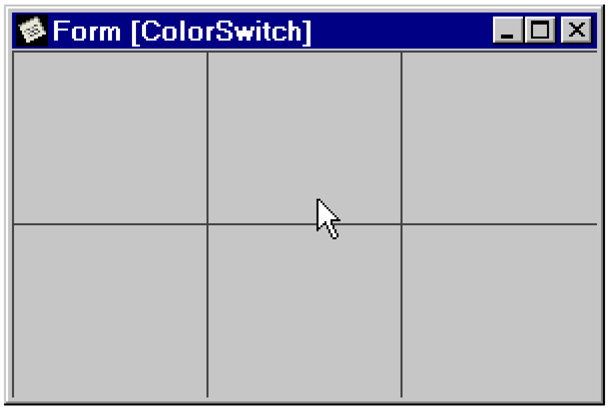
All containers have a layout manager which controls the appearance and placement of components in the container. The default layout manager is Border layout, which divides the container into five sections (one large center section and a smaller section on each of the four sides). We'll switch the layout manager to Grid layout, in which the container is divided into a simple grid.

To switch the layout manager to Grid layout:

1 Right-click on `BorderLayout` in the Component Inspector.

2 Select **Set Layout | DesignGridLayout** from the context menu.

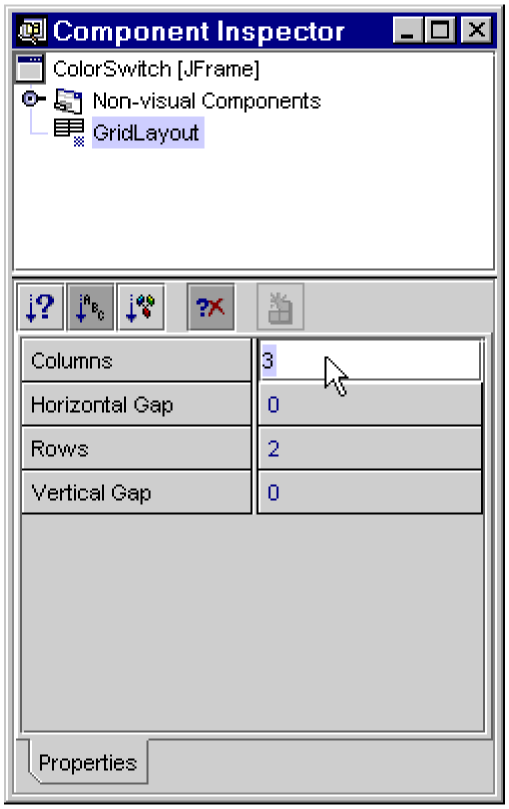
The Form Editor window should now display a two-row, three-column grid.



If you click on the `GridLayout` node in the Component Inspector, you will see that the properties listed in the lower panel of the Component Inspector (the Property Sheet pane) have changed to reflect the properties of the new layout manager. We'll now modify these properties to make the grid two rows by one column.

To change the grid:

1 Make sure the `GridLayout` node in the Component Inspector is selected.



Columns	3
Horizontal Gap	0
Rows	2
Vertical Gap	0

2 Click on the value in the `Columns` property (`3`), enter `1` in its place, and press ENTER (RETURN).

Important: When *typing* new values in a property sheet, you must always press ENTER to confirm the entry. Otherwise, the property will revert to its previous value.

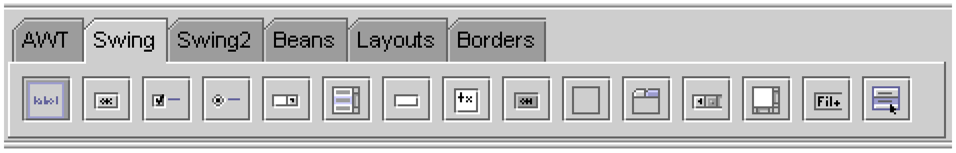
Adding components and setting their properties

Next we'll add the visual components – a label and button in this case – to the application by using the Component Palette, the toolbar in the upper right corner of the Main Window.

To add the label:

1 Click on the **Swing** tab (to display JFC icons).

2 Click on the **JLabel** icon.

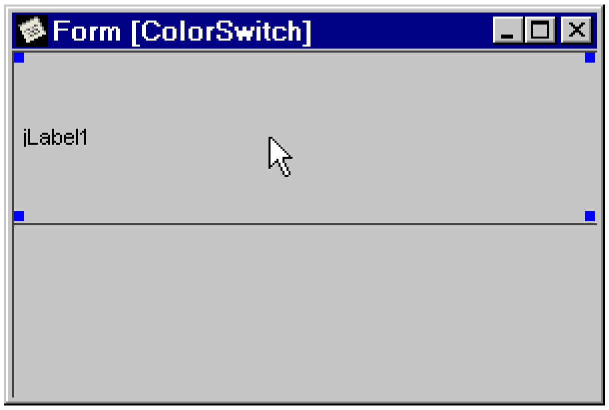


3 Click anywhere on the Form window.

Tip: To identify a toolbar icon, hold your mouse cursor over it, and a tool tip label will appear with its name.

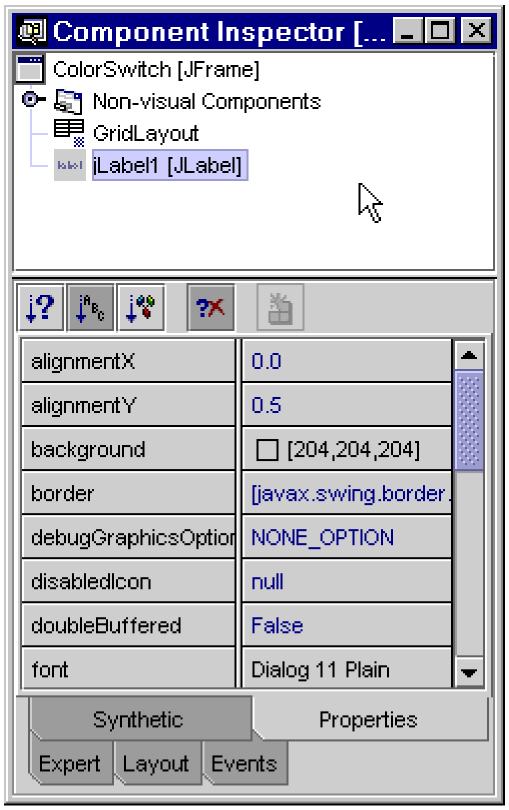
After adding the label to the Form Editor, you'll see a lot of changes:

- In the Form Editor window, the label is displayed with the default text `jLabel1` and blue squares in its corners (indicating it is the current selected object).



- A new node called `jLabel1 [JLabel]` appears in the Component Inspector.

- The Property Sheet pane displays the label's properties.



alignmentX	0.0
alignmentY	0.5
background	☐ [204,204,204]
border	[javax.swing.border.
debugGraphicsOptior	NONE_OPTION
disabledIcon	null
doubleBuffered	False
font	Dialog 11 Plain

- New source code is added to the file in the Editor window.

To add the button:

1 Click on the **JButton** icon.

2 Click anywhere on the Form window.

You'll see more changes in the Form Editor window, Component Inspector, and Editor window.

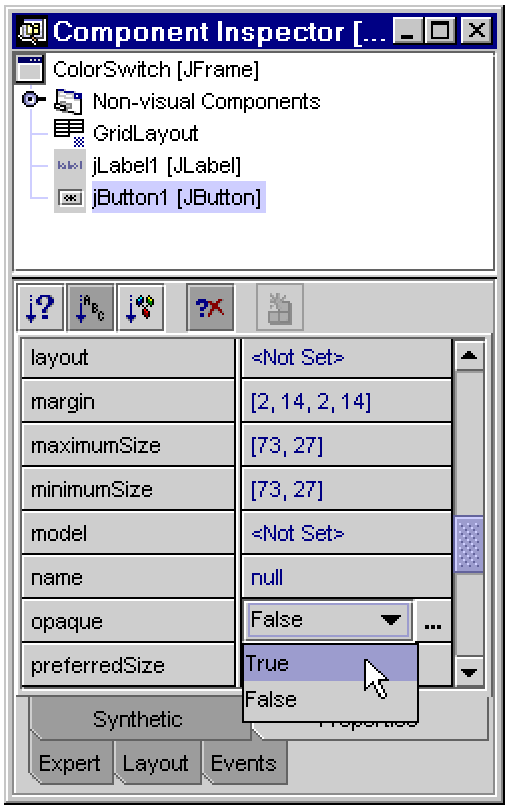
Now we'll use the Property Sheet pane in the Component Inspector to remove the default text (`jButton1`) from the label and change the label text on the button.

To make these property changes:

1 Select the `label1[Label]` node in the Component Inspector (if it isn't already highlighted).

2 Select the `opaque` property and then use the drop-down list to change its value to `True`.

> **Note:**In v. 1.3 of the Java™ 2 SDK, `opaque` is an expert property. So if you are using version 1.3 or higher of the Java™ 2 SDK, you must click the **Expert** tab at the bottom of the Component Inspector to find the `opaque` property.



layout	<Not Set>
margin	[2, 14, 2, 14]
maximumSize	[73, 27]
minimumSize	[73, 27]
model	<Not Set>
name	null
opaque	False
preferredSize	

3 Switch back to the **Properties** tab and scroll down to the `text` property, delete `jLabel1`, and press ENTER.

4 Now select the `jButton1[Button]` node.

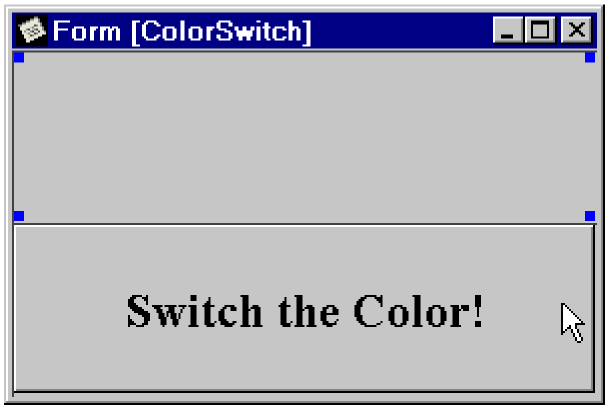
5 For the `text` property, type <u>`Switch the Color!`</u> and press ENTER.

6 Select the `font` property, and then press the **...** button that comes up to invoke the custom

Property Editor dialog.

7 In the dialog, select the **Serif** font, **Bold** font style, and **24** for font size, and click **OK**.

Your form should now look like this:



Automatically generating event code

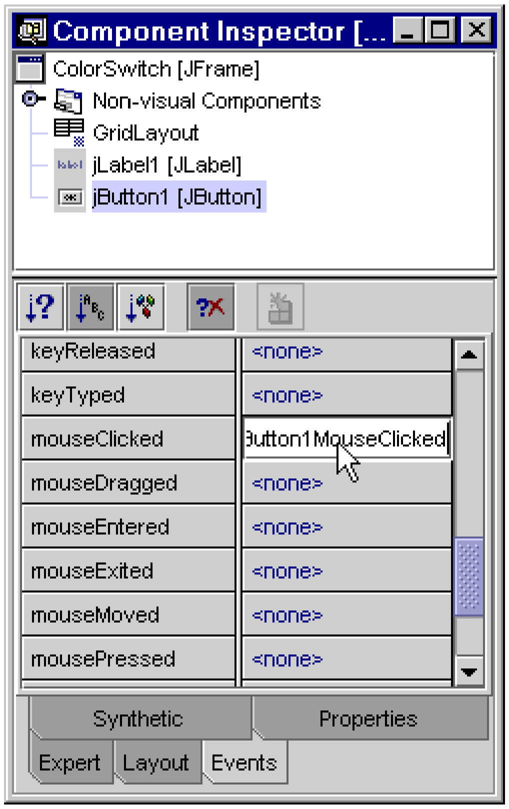
Now that we've created some visual objects, it's time to do something with them. So we'll add an **event** to the program to give functionality to the button.

To add the event to the button:

1 Select the `jButton1` node in the Component Inspector (if it isn't selected already).

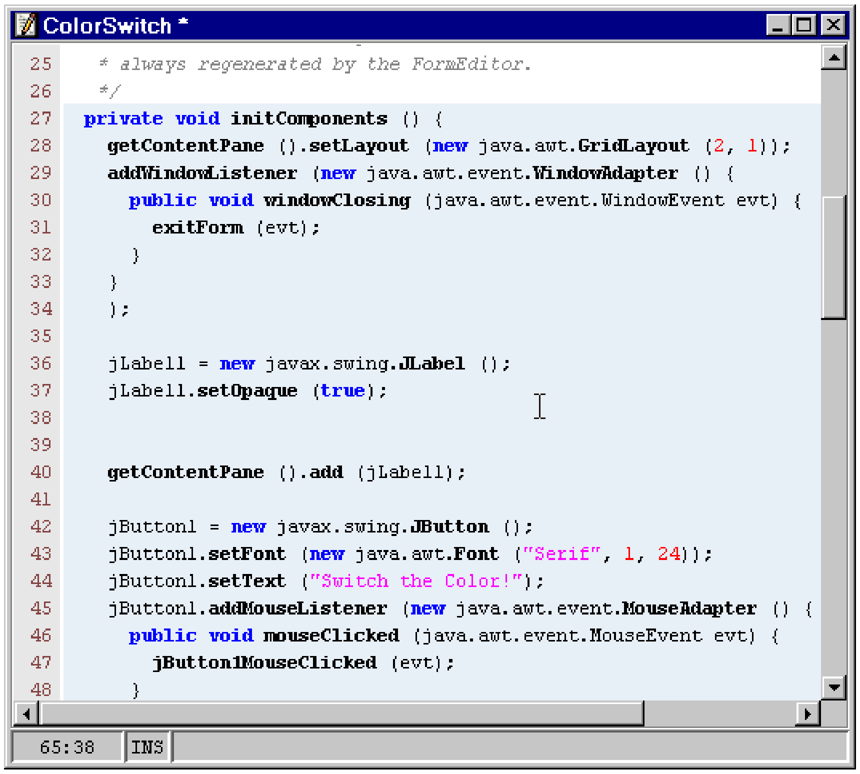
2 Click the **Events** tab in the Property Sheet pane.

3 Select the `mouseClicked` property and press ENTER.



keyReleased	<none>
keyTyped	<none>
mouseClicked	3utton1MouseClicked
mouseDragged	<none>
mouseEntered	<none>
mouseExited	<none>
mouseMoved	<none>
mousePressed	<none>

If you look at the Editor window, you will see that new code has been automatically generated – namely the listener code (`jButton1.addMouseListener`) and event method (`jButton1MouseClicked)`).



```
     * always regenerated by the FormEditor.
     */
    private void initComponents () {
      getContentPane ().setLayout (new java.awt.GridLayout (2, 1));
      addWindowListener (new java.awt.event.WindowAdapter () {
        public void windowClosing (java.awt.event.WindowEvent evt) {
          exitForm (evt);
        }
      }
      );

      jLabel1 = new javax.swing.JLabel ();
      jLabel1.setOpaque (true);


      getContentPane ().add (jLabel1);

      jButton1 = new javax.swing.JButton ();
      jButton1.setFont (new java.awt.Font ("Serif", 1, 24));
      jButton1.setText ("Switch the Color!");
      jButton1.addMouseListener (new java.awt.event.MouseAdapter () {
        public void mouseClicked (java.awt.event.MouseEvent evt) {
          jButton1MouseClicked (evt);
        }
```

Adding event handler code

Now that we have created event code (for when the mouse is clicked on the button), we need to add the event handler code telling the program *what* to do when the mouse is clicked. When the IDE generates the event code, the Editor window scrolls to the point where you should add the handler code (it's labeled `// Add your handling code here:`).

Before we enter the event handler code, we need to declare a variable. In the bottom of the source code, just after the comment, `//End of variables declaration`, add your own:

```
private java.awt.Color currentColor = java.awt.Color.lightGray;
```

Note: You can only type in white areas. The shaded text is uneditable.

At the point in the Editor window where the IDE tells us to add handling code, we will type the following:

```
if (currentColor == java.awt.Color.lightGray)
  currentColor = java.awt.Color.gray;
else if (currentColor == java.awt.Color.gray)
  currentColor = java.awt.Color.black;
else
```

```
        currentColor = java.awt.Color.lightGray;
      jLabel1.setBackground (currentColor);
```

Compiling and running your program

Your program should be ready to compile and run.

To compile ColorSwitch:

1 Make sure that it is active in the Form Editor window or Editor.

2 Select **Compile** from the **Build** menu or press F9.

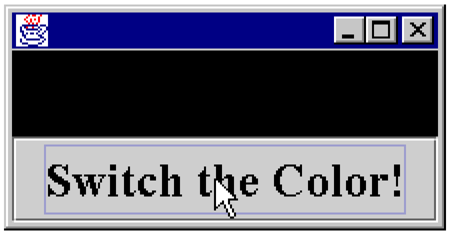
If compilation is not successful, any errors will be noted in the Output Window. Press ENTER or double-click on the error line in the output to jump to the error in the Editor. Once you have corrected any errors and successfully compiled, the Status Bar in the Main Window will read `Successfully compiled ColorSwitch`. Now you can run the application.

To run ColorSwitch:

1 Make sure that it is active in the Form Editor window or Component Inspector.

2 Select **Execute** from the **Build** menu or press CTRL+F9.

Assuming there are no execution errors, the IDE will switch to the Running Workspace, which means that the Execution View and Output Window will open along with our application. Each time you click the button, the color of the label should switch, cycling through light gray, gray, and black.



Congratulations! You have just written and run your first program in Forte for Java Community Edition 1.0. Now you can move on to the next three chapters for more information.

- Chapter 4, Developing Java Applications, takes you through the development process in greater detail, showing you how to create, run, and debug classes.
- Chapter 5, Developing Visual Classes tells you everything you need to know about creating visual forms and creating JavaBeans™ Components.
- Chapter 6, Using the IDE, is a guide to all of the aspects of the programming environment, including the Editor and other windows, workspaces, templates and the many customization options.

Chapter 4

Developing Java Applications

This chapter explains how to use the Forte for Java Community Edition 1.0 IDE to create and run applications. Specifically, we'll show you how to generate and edit Java objects and code using the Editor, Object Browser, Explorer, and templates. Finally, we'll explain how to debug, compile and execute objects with the IDE.

The Explorer and its Property Sheet pane

Most development tasks can be managed in the IDE's Explorer window, which appears below the Main Window when you first launch the IDE. The Explorer provides easy access to files, IDE settings, and various services and extensions. It consists of a pane which provides a tree view of files and a pane which displays the property sheet of the node selected in the tree. For more information on the Explorer, see "The Explorer" on page 118.

The Property Sheet pane can be displayed or hidden by pressing the button in the Explorer's toolbar. Depending on the type of node selected, the property sheet displays available Java properties (both writable and read-only) and other configuration options. For more information on working with property sheets, see “Property sheet” on page 119.

Creating new classes

In Forte for Java Community Edition 1.0, you create new classes with templates. The template serves as a skeleton for the class and includes basic source code for that class. If you prefer to write all of the code yourself, you can choose the `Empty` template, and a file will be generated with the only code being the name of the package where you have created the template. A wide range of templates come with the IDE, and you can also create your own. For more on templates see “Using templates” on page 148.

There are several ways to create a new class.

To create a class from the Main Window:

1 Select **New From Template** in one of the following ways:

- from the **File** menu
- from its toolbar icon
- using the CTRL+n keyboard shortcut

2 Select the type of template (e.g. **Classes**) by clicking on one of the tabs in the Templates dialog.

3 Click on the icon of the template you want and select **OK** or just double-click on the template icon. (Type just the base name without `.java` or any other extension).

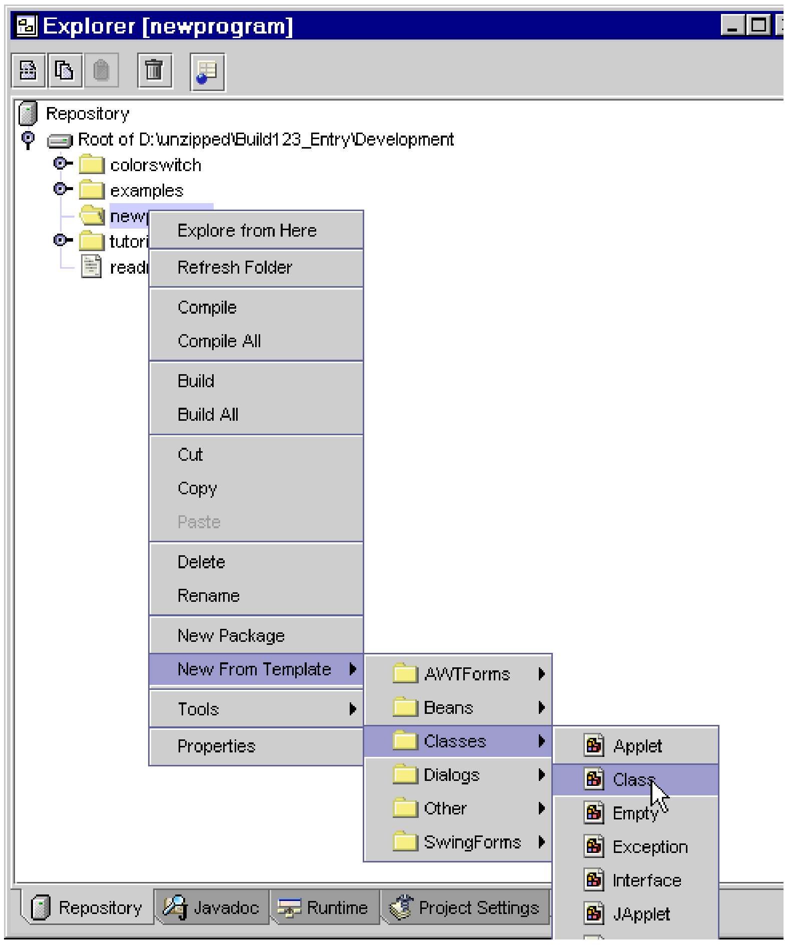
4 The Instantiate Template dialog will appear. Select a location in the tree (package) where you want to place the class, type a name for it in the **Object Name** field, and click **OK**.

To create a class from the Explorer

1 If the Explorer isn’t open, select **Open Explorer** from the **File** menu or toolbar or press CTRL+O.

2 Find the package (marked with a folder icon) under the `Repository` tab in the Explorer where you want to place the class and right-click on it to bring up its context menu.

3 Select **New From Template**, the template type from the first submenu, and then the template itself from the second submenu.



4 In the **New** dialog that appears, type the name of your new object and click **OK**.

Once you have created the class, the Editor window (or a tab in the Editor window if the Editor is already open) will open up and display the skeleton code for that class already generated. The new class will also be automatically added to the Explorer's tree and the Object Browser (see "Object Browser" on page 57).

Adding a file to the IDE

The Forte for Java Open File feature provides the ability to open existing (i.e. not created in the IDE) source files in the IDE straight from a file chooser without the need to first mount directories in the Repository or navigate to them in the Explorer.

To open a file in the IDE:

1 Select **Open File** from the first toolbar or the **File** menu in the Main Window. A file chooser will

appear asking you which file you wish to open.

2 Browse to the directory on disk where your file is (for example, `MyFirstClass.java`), select it, and press **Open**.

   If the file is already accessible in the Repository, then the file will open (usually in the Editor window) immediately. If not, you must:

3 Decide which directory to "mount" – that is, which directory containing the source file should correspond to the default package in Java. A dialog will appear with a list of possible choices for the containing directory; you must select one and press **Mount**.

If you are opening a Java source file, the IDE tries to determine the correct directory to mount by looking through the source file and trying to find a package declaration. For example, if you are opening `C:\mysources\com\mycom\Foo.java`, and this file begins with the declaration `package com.mycom;`, then `C:\mysources` will be selected as the default. If there is no package declaration, the directory directly containing the source file will be the default to mount.

You can override the default, but be sure that you choose the right mount point. If you do not choose the correct mount point, you will not be able to work with the file (your package declaration will be invalid, you won't be able to compile or execute the source, debugging won't work, etc.). When opening other types of files (such as GIF images, HTML, or resource property files) that do not have package declarations, you must be sure to mount the correct directory – look at the bottom of the dialog to see what "package" will be used for the file. The choice will affect any Java sources that are in the same mounted directory. If you are using classloader-based resource loading (i.e. based on an abstract resource name such as `/com/mycom/myImage.gif`), such as when creating a JAR file for distribution, then the resource names and JAR manifest entry names will be relative to the mounted directory as well.

Files which cannot be opened for editing will typically just be displayed in their own Explorer window. You may also select ZIP or JAR archives with Open File. When you do, they are immediately mounted in the Repository and an Explorer window is opened on their contents to make it easy to browse archives.

Note: Mounted JAR archives are read-only.

Editing Java sources

Forte for Java's Editor displays code generated by other parts of the IDE and provides versatile features to simplify and accelerate manual coding.

The Editor and its layout

The Editor contains source code that is syntactically colored. Different colors signify different text properties. For example, by default all keywords are shown in blue and all comments in light gray. **Guarded text** generated by the Form Editor has a blue background by default and cannot be edited.

The bottom of the Editor window has one or more tabs used to view different documents. From each of these tabs, the document can be saved, closed, docked or undocked, and cloned. For more details on managing windows, see "Window management" on page 130.

There are many built-in keyboard shortcuts for navigation and editing, as well as abbreviations for commonly used keywords. See below for more information.

Integration with other components

The Editor is integrated with the following parts of the IDE:

- **Form Editor** – All changes made in the Form Editor window are reflected in the source code in the Editor window.
- **Explorer** – Changes in properties and creation of new classes, methods, and variables, etc. are reflected in the Editor window. There are several ways to open a file in the Editor, including pressing ENTER, selecting **Open** from the right-click popup menu, and double-clicking on an icon representing a class, method, variable, etc.
- **Debugger** – When a program stops during execution, the Editor automatically jumps to the breakpoint where the code was interrupted. You can toggle breakpoints with CTRL+F8.
- **Compiler** – If there is a compilation error, the cause of the error will be highlighted in red in the Output Window; press ENTER or double-click on that line to jump to the buggy code in the Editor.

For more detail on how Editor features, see "Editor" on page 122.

Editor Settings

The Editor has default font size, style, and color settings. You may customize these settings separately for Java, HTML, and plain text under their respective subnodes under `Project Settings / Editor Settings` in the Explorer. "Editor Settings reference" on page 182 lists these properties.

Compiling Java sources

Forte For Java Community Edition 1.0 offers a wide array of compilation options, from different ways to invoke the **Compile** command to the ability to use different compilers and set a specific compiler for each class.

Note: When you choose the Compile (or Compile All, Compile Project, Build, Build All, or Build Project) command for an object, the IDE (consistent with Java conventions) automatically compiles the first file it finds with the same name and package. Therefore, if you have two files with the same file name and package hierarchy mounted in the Repository, the file in the first package listed will be compiled automatically, even if you choose the Compile command with the second package selected

Compiling single classes

You can compile an object in the active Editor window tab or if selected in the Explorer by:

- selecting **Build | Compile** from the main menu; or
- clicking on the **Compile** icon on the main toolbar; or
- pressing F9; or
- right-clicking on the object in the Explorer and selecting **Compile** from the popup menu.

Compiling packages

There are several options for compiling packages, all available from the **Build** menu and toolbar on the Main Window and the popup menu for packages in the Explorer:

- Selecting **Compile** when a folder is selected compiles all sources in that folder which have been modified since they were last compiled or that have not been previously compiled.
- Selecting **Compile All** does this recursively on a folder and all its sub-folders.
- Selecting the **Build** command (also available with keyboard shortcut ALT+F9) is slightly different from compiling in that it forces re-compilation of *all* sources in a folder, *whether they are current or not*. Use this option when you wish to be sure that all of your code can compile together.
- Selecting **Build All** recursively builds a folder and all sub-folders.

Deleting .class files

The **Build** menu also has the commands **Clean** and **Clean All** which delete compiled classes. **Clean** deletes all `.class` files in the selected package and **Clean All** recursively deletes all `.class` files in the selected package and its sub-packages.

Switching compilers

A default compiler is set for each class. If you would like to use a different compiler (or a different configuration of a compiler) for that class, you can specify one on the class's property sheet.

To switch compilers for a class:

1 Select the object under the **Repository** tab in the Explorer.

2 Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).

3 Click on the **Execution** tab in the Property Sheet window (or pane).

4 Rotate through the compiler types by double-clicking on `Compiler`, or click on the `Compiler` property's value and choose from the pull-down menu.

By default, there are two choices for Java sources: using Javac internally (in the same VM as the IDE), and running Javac externally. Other types of files have different choices.

Disabling compilation for a class

If you have a source in the Repository which you specifically do not want to be subject to compilation, you can disable compilation for that class.

To disable compilation for a class:

1 Select the object under the **Repository** tab in the Explorer.

2 Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).

3 Click on the **Execution** tab in the Property Sheet window (or pane).

4 Click on the `Compiler` property's value and choose `(do not compile)` from the pull-down menu.

Configuring compilers

It is also possible to customize the command-line template for the executable compiler, thus affecting the way the compiler is invoked. For more information, see “Adding and modifying service types” on page 72.

Running Java classes

Java applications may be executed in several ways.

To execute a Java application:

1 Make sure that the Java object is executable (i.e. that it either has a `main()` method or is a subclass of `Applet` or `JApplet`).

2 Right-click on it in the Explorer and select **Execute** from the popup menu.

Alternately, you can select the Java object in the Editor window and then run it one of the following ways:

- Select the **Execute** icon on the Main Window.
- Select **Build | Execute** from the Main Window.
- Use the keyboard shortcut CTRL+F9.

When executing, the Java class is (by default) first compiled. Assuming compilation completes successfully, the IDE then switches to the Running Workspace (though you may configure it to do otherwise by going to `Project Settings / Execution` in the Explorer and changing the `Workspace` property). See “Workspaces” on page 130 and “Customizing workspaces” on page 156 for more information.

Execution categories and executors

You can run typical Java applications using either internal or external execution, each of which have their advantages and disadvantages. Applets can be run using Applet Execution – see “Applet viewer settings” on page 48 for more information. Other execution categories can be installed by extension modules.

External Execution

Most applications use external execution, and it is set in most of the templates that come with the IDE.

A new virtual machine (VM) is invoked for executing the application. This enables you to run applications which require a special VM or need to do operations that aren't possible with internal execution (see below). You can specify the VM executable (such as `java.exe`) and complete command line parameters together with class path settings for the application. External execution also protects the IDE from application crashes and delays.

Internal (Thread) Execution

An application run using internal execution runs inside the Forte for Java IDE. This brings the advantages that the application can modify the IDE itself and be loaded faster. But it imposes at least two restrictions on the executed application. The application cannot install its own `URLStreamHandlerFactory` or `SecurityManager` (so you cannot run RMI applications, for example). In addition, if the executed application crashes, the IDE crashes with it. Go to the `Examples` directory in the Repository to look at some samples of internal execution applications.

Note: Some applications, such as startup routines (in the `Global Settings / Startup` folder of the Explorer) *require* the use of internal execution because they're intended to modify the IDE itself.

Other execution categories

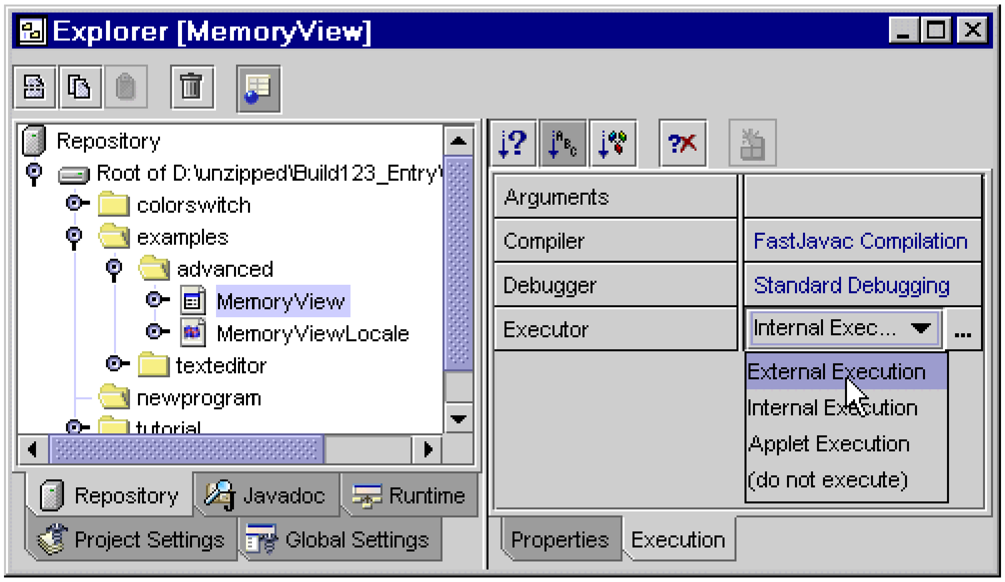
Other execution categories tailored for specific types of applications are installed by various modules, such as RMI and JSP.

Setting execution

The execution category (e.g. external, internal, or applet) is set for each separate object. When you set execution, you choose from a list of "executors", each of which represents a specific configuration (with parameters such as the path to Java, the working directory, and other arguments) of an execution category. There can be multiple executors for a given execution category, though the IDE comes with only one for most categories. See "Adding and modifying service types" on page 72 for more information.

To switch an object's executor:

1. Select the object under the `Repository` tab in the Explorer.
2. Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).
3. Click on the **Execution** tab in the Property Sheet window (or pane).
4. Rotate through the executors by double-clicking on `Executor`, or click on the `Executor` property's value and choose from the pull-down menu.



Arguments	
Compiler	FastJavac Compilation
Debugger	Standard Debugging
Executor	Internal Exec...

Disabling execution for a class

If you have a source in the Repository which you specifically do not want to be subject to execution, you can disable execution for that class.

To disable execution for a class:

1. Select the object under the **Repository** tab in the Explorer.
2. Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).
3. Click on the **Execution** tab in the Property Sheet window (or pane).
4. Click on the `Executor` property's value and choose `(do not compile)` from the pull-down menu.

Configuring external executors

It is also possible to customize the command-line template for the executor, thus affecting the way the executor is invoked. For more information, see "Adding and modifying service types" on page 72.

Passing command-line arguments to executed applications

To pass command-line arguments to executed Java applications:

1. Select the object in the Explorer.
2. Choose **Build | Set Arguments** from the main menu.

3 Enter the arguments in the dialog, separated by spaces.

or

1 Select the object and open its property sheet (by right-clicking on the node and selecting **Properties** or clicking the Toggle Property Sheet icon).

2 Click the **Execution** tab, type the argument in the `Arguments` property, and press ENTER (if you don't press ENTER, the changes won't be kept).

Note: This applies only to application arguments, not to Java virtual machine arguments (which must be configured on the object's executor – see "Adding and modifying service types" on page 72).

Execution Settings node

Under the **Project Settings** tab in the Explorer is the `Execution Settings` node, where you can configure the IDE's behavior when running applications. The options include whether to automatically compile applications before execution, whether to create a new output tab for each executed. See "Execution Settings reference" on page 185 for more information.

Applet viewer settings

The IDE allows you to choose which viewer to use when running applets. You can use either:

- Sun's JDK utility AppletViewer, which is set by default; or
- an external viewer such as Netscape Navigator or Microsoft Internet Explorer.

For security reasons, internal execution is not allowed for applets.

To change the default viewer:

1 Go to `Project Settings / Executor Types / Applet Execution / Default` in the Explorer.

2 Go to the object's property sheet (by right-clicking on the node and selecting **Properties** or clicking the Toggle Property Sheet icon).

3 Click on the `External Viewer` property and type in the path and name of the browser or applet viewer (and press ENTER)

4 In the same custom property editor, add any startup arguments that you require for the applet viewer.

To set up a viewer other than AppletViewer:

1 Right-click on `Applet Execution` under `Project Settings / Executors / Applet Execution` in the Explorer.

2 Select **New | Applet Execution Service (by prototype)** or **New | Applet Execution Service (fresh)** from the context menu. (The first option gives you a replica of the current default applet executor type and the second gives you a generically configured type).

3 A new node labelled `Applet Execution` (with a number in parentheses to give it a unique name) will appear. Select it and, if you wish, modify its `name` property.

4 Follow steps 2 through 4 from the procedure for changing the default applet viewer.

You can now use this new executor for particular objects.

Debugging Java classes

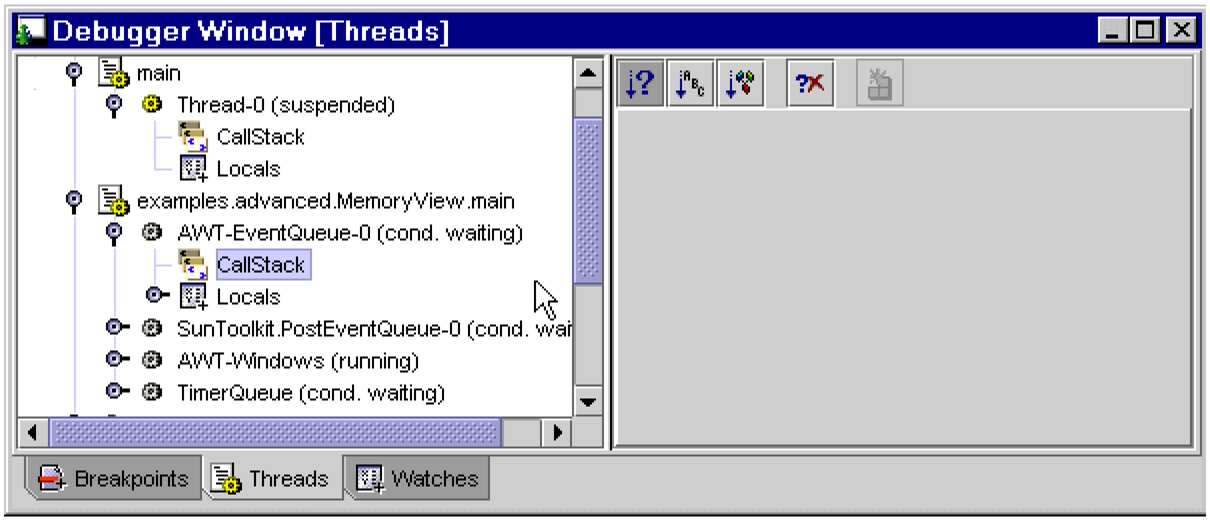
The Debugger can be used to present "snapshots" of the system state during execution. By placing breakpoints at key positions throughout your source code, the Debugger can halt at these points and display details of the current environment at that point in the source. You can effectively step through your code, monitoring execution as it occurs. You can also connect the debugger to an already running process.

Forte For Java Community Edition 1.0 provides support for two debuggers – the standard "Tools" Debugger and the new JPDA Debugger. The two debuggers work in the same way except that the JPDA Debugger provides a few extra features. If you are using a version earlier than 1.3 of the Java™ 2 SDK, you may need to install Sun's support. See "JPDA Debugger" on page 55 for more information.

Debugger Window

The Debugger Window is a three-tabbed display with tabs for **Breakpoints**, **Threads**, and **Watches**. In the right half of the window is the property sheet pane which displays the properties and their current values for the selected node in the left pane.

Debugger Window with the Thread Group tab selected


Debugger Window [Threads]
main
Thread-0 (suspended)
CallStack
Locals
examples.advanced.MemoryView.main
AWT-EventQueue-0 (cond. waiting)
CallStack
Locals
SunToolkit.PostEventQueue-0 (cond. wai
AWT-Windows (running)
TimerQueue (cond. waiting)
Breakpoints
Threads
Watches

Breakpoints

The **Breakpoints** tab simply lists the currently set breakpoints, showing the class name, and the line number or method on which the breakpoint has been set.

To add a new breakpoint to your code:

1 Position the cursor at the desired line in the Editor window.

2 Select the **Debug | Toggle Breakpoint** menu or toolbar item from the Main Window or use the keyboard shortcut CTRL+F8.

The current line will be highlighted blue to indicate that the breakpoint has been set.

or

1 Select **Add Breakpoint** from the **Debug** menu or toolbar to invoke the Add Breakpoint dialog.

2 Choose the type of breakpoint (either “line” or “method”) from the drop-down list.

3 Enter the settings (class name and line number or method name).

Optional breakpoint settings

If you set the breakpoint using the **Add Breakpoint** command, you have further options:

- If you want to be notified in the Output Window when the breakpoint is reached, check **Print text** in the Add Breakpoint dialog. You can also set the text to be printed using a combination of plain text and these self-explanatory substitution codes: `{lineNumber}`, `{className}`, and `{threadName}`. In addition, you can use curly braces and a dollar sign to create a substitution code for a watch (e.g. `{$mywatch}`).

- Checking **Stop Debugging** suspends the debugging session (all threads) when the breakpoint is reached.

You can also set these options (and later change them, if you wish) in the property sheet for the breakpoint in the Debugger Window.

To remove a breakpoint:

1 Position the cursor on the line in the Editor window where the breakpoint has been set.

2 Select **Debug | Toggle Breakpoint** from the Main Window or use the keyboard shortcut CTRL+F8.

Breakpoints can also be removed directly from the Debugger window by right-clicking on a listed breakpoint and selecting **Delete** from the popup menu or by selecting the breakpoint and pressing DELETE on the keyboard.

Threads

The **Threads** tab displays all thread groups in the current debugging process. These thread groups are expandable hierarchies; each group containing other thread groups or single threads, which in turn contain `CallStack` and `Locals` nodes.

When a thread is suspended:

- The `CallStack` node can be expanded to show the current hierarchy of method calls made during execution.
- The `Locals` node displays local variables and their current values in the context of the current thread. You can expand these nodes to see the object sub-structure.

If the process you are debugging has more than one thread, all threads and thread groups appear in the **Threads** tab showing a thread name and current status (such as “running”, “at breakpoint”, “cond. waiting”and “suspended”). Suspended threads and threads at breakpoint display all “current” system information.

The Debugger Window displays the following properties for each running thread:

- `Name` – thread name (according to the thread class)
- `State` – status of the thread, such as `Running`, `Cond. waiting`, etc.
- `Class` – name of the class in which the thread is suspended
- `Method` – name of the method in which the thread is suspended
- `Suspended` – If `True`, the thread is suspended

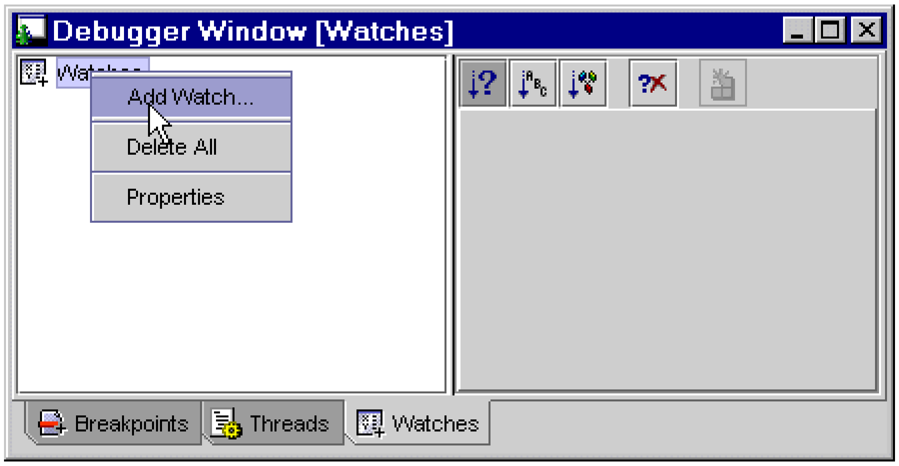
Watches

The **Watches** tab lists all currently set watches or values of local variables. By watching a variable, you can monitor its value at various stages of execution.

To watch a variable:

1 Select **Add Watch** from either

- the **Debug** menu on the Main Window; or
- the popup menu of the root `Watches` item on the **Watches** tab of the Debugger Window; or
- or from the popup menu of a variable you have selected in the Editor.



A dialog box requesting the name of the variable to watch will open. Once you have entered the name of a variable in your source and clicked **OK**, it will be listed in the `Watches` tree.

2 Click on this item in the `Watches` tree to select it and display its property sheet.

3 Continue running the application and watch the variable change.

Tip: You can also set a watch by double-clicking the variable in the Editor window, right-clicking, and then selecting **Add Watch** from the context menu.

Fixed watches

You can also take "snapshots" of the watch in various contexts in the debugged code by adding fixed watches.

To add a fixed watch:

◊ Right-click on the watch and select **Create Fixed Watch** from the context menu.

A new node will appear in the Watches tree showing the current value of the watched variable. This value will be frozen, even as the watched variable changes. You can add multiple fixed watches for the same variable in different contexts.

The debugging session

To initiate a debugging session:

1 Set a breakpoint and then select **Debug | Go** from the Main Window (or press F5). (If you are debugging a GUI application or another looped application, it is not necessary to set a breakpoint.)

By default, the IDE switches to the Debugging Workspace (to configure it to do otherwise, see “Customizing workspaces” on page 156), where the Debugger Window, the Editor with the source being debugged, and the Output Window all open up. The Output Window is split vertically, with Debugger output on the right and any output from the application being debugged on the left. When the Debugger reaches a breakpoint in your source, that breakpoint is highlighted pink. The pink line will move through your source as you code as you step through its execution.

Debugging can also be initiated by selecting the **Trace Into** command, which causes the Debugger to stop on the first line after the main method.

2 Once execution has halted (whether on a breakpoint or just after the main method), use the **Trace Into**, **Trace Over** and/or **Go** menu or toolbar items under the Main Window **Debug** menu (or the keyboard shortcuts F7, F8, and F5, respectively) to proceed.

Trace Into steps into the method at which the Debugger is currently stopped if there is a method call on that line and breaks at the start of the called method, allowing you to observe execution incrementally. If there is no method call on the current line, then it behaves like **Trace Over**.

Trace Over executes the current statement without breaking and stops at the next statement.

Step Out (which you can select from the Main Window) halts execution after the current method finishes and control passes to the caller.

Go resumes execution, which continues until it reaches the next breakpoint or the end of the application.

Finish Debugger ends the current debugging session.

By stepping through your code like this, you can monitor whatever parts of the system you choose during execution of the code.

Suspending and resuming debugging

The **Debug** menu and toolbar also have **Suspend All** and **Resume All** options, which allow you to “pause” execution at any time and then continue from the point execution was suspended.

To suspend selected threads or thread groups:

1 Under the **Thread Group** tab in the Debugger Window, select the nodes of those threads or thread groups (using SHIFT to select multiple consecutive nodes and CTRL to select various non-consecutive nodes).

2 Right-click on one of the selected nodes and select **Suspend** from the context menu.

To suspend all threads:

◊ Select **Suspend All** from the **Debug** menu or toolbar; or

◊ Right-click on the root **Thread Group** node in the Debugger Window and select **Suspend** from the context menu.

When a thread is suspended, the Debugger window displays all current information for the thread.

Likewise, you can resume any or all of the suspended threads. Select **Resume All** from the **Debug** menu to resume execution of all threads. To resume execution for threads individually, right-click on the thread or thread group and select **Resume** from the context menu.

Changing the current thread

The current thread is set automatically when a breakpoint is reached. You can also change it manually.

To change the current thread:

◊ Under the **Thread group** tab in the Debugger Window, right-click on the node of the thread you would like to switch to and select **Switch to thread** from the context menu.

Connecting the Debugger to a running process

To connect the Debugger to an already-running virtual machine:

1 When launching the process, enter `-Xdebug` in the Java virtual machine's parameter list (after `-classic` when running on HotSpot) and note the agent password.

2 Select **Connect** from the **Debug** menu or toolbar to invoke the Connect to Running VM dialog.

3 Enter the host name and agent password in the dialog.

After clicking **OK**, the Debugger will connect to the running VM, and you will be able to see all threads as if you were debugging locally. If you have source code for the debugged application and you set a breakpoint in the source code, the Editor will be opened with the breakpoint line highlighted in the source.

Setting the debugger

The debugging category (e.g. applet, default, or JPDA) is set for each separate object in the IDE. When you set debugging, you choose from a list of "debugger types", each of which represents a specific configuration of a debugger (with parameters such as the path to Java, the working directory, and other arguments). There can be multiple debugging types for a given debugging category, though the IDE comes with only one for each category. See "Adding and modifying service types" on page 72 for more information.

To switch an object's debugging type:

1. Select the object under the `Repository` tab in the Explorer.
2. Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).
3. Click on the **Execution** tab in the Property Sheet window (or pane).
4. Rotate through the debugging types by double-clicking on `Debugger`, or click on the `Debugger` property's value and choose from the pull-down menu.

Disabling debugging for a class

If you have a source in the Repository which you specifically do not want to be subject to debugging, you can disable debugging for that class.

To disable debugging for a class:

1. Select the object under the **Repository** tab in the Explorer.
2. Go to the object's property sheet (by right-clicking on the node and selecting **Properties**, clicking the **Toggle Property Sheet** icon, or pressing ALT+1).
3. Click on the **Execution** tab in the Property Sheet window (or pane).
4. Click on the `Debugger` property's value and choose `(do not compile)` from the pull-down menu.

Configuring debuggers

It is also possible to customize the command-line template for the debugger, thus affecting the way the debugger is invoked. For more information, see "Adding and modifying service types" on page 72.

JPDA Debugger

Forte for Java, Community Edition 3.0 also supports the new Java Platform Debugger Architecture (JPDA) Debugger. If you are running Java™ 2 SDK v. 1.3 or later for Windows or Solaris, Sun's JPDA support is already installed. If you are running an earlier version, you will need to download and install that support.

Installing JPDA

To install Sun's JPDA support for Windows:

1 Download the JPDA support files from Sun at http://java.sun.com/products/jpda/.

2 Unzip `jpda1_0-win.zip` to a directory on your hard drive, such as `C:\jpda`.

3 Add the `C:\jpda\bin` (or wherever you put it) folder to the system path in the IDE startup script `forte4j.bat` or `forte4j_nt.bat` (whichever one you use). The result should be a line in the script that looks like: `set PATH=C:\jpda\bin;%PATH%`

4 Copy the file `C:\jpda\lib\jpda.jar` to the `%JAVA_HOME%;\jre\lib\ext` folder.

5 Restart the IDE.

To install Sun's JPDA support for Solaris:

1 Download the JPDA support files from Sun at http://java.sun.com/products/jpda/.

2 Unzip `jpda1_0-solsparc.zip` to a directory on your hard drive, such as `/jpda`.

3 Add the `/jpda/bin` folder to the system path in the IDE's startup script `forte4j.sh`. The result should be a line that looks like: `PATH=/jpda/bin:$PATH`

4 Copy the file `/jpda/lib/jpda.jar` to the `%JAVA_HOME%/jre/lib/ext` folder.

5 Restart the IDE.

Setting the JPDA Debugger

To use JPDA Debugging, you will need to set the debugger for the class you're debugging to JPDA Debugging.

To set the debugger to JPDA for a class:

1 Select the class to be debugged under the **Repository** tab in the Explorer.

2 Under the **Execution** tab in the property sheet, set the `Debugger` property to `JPDA Debugging`.

Tip: If you generally prefer to use JPDA debugging, you can change the debugger in the templates you use to create new objects. See "Modifying existing templates" on page 151.

Additional JPDA debugging features

The JPDA module adds the following debugging functionality:

- **Watch Expressions** – you can define an expression as a watch, such as `(myClass.y * myClass.width) + myClass.x` and the expression will be evaluated on the fly while debugging and displayed for you.
- **Break on exceptions** – when an exception is encountered, a breakpoint is created and execution stops. You can specify the kind of exceptions that will trigger a break.
- **Break on classes** – the debugging process stops when a class is reached or left. You can specify a filter to determine on which classes the break is to occur.
- **Break on threads** – the debugging process stops on the start or death of a thread.
- **Break on variables** – the debugging process stops on variable access or modification. You must specify a class and its variable name.

Object Browser

The Object Browser gives you a Java-oriented perspective on your classes, allowing you to view a hierarchy of source files filtered in the way that you specify.

To open the Object Browser:

◊ Click on the **Browsing** workspace tab; or

◊ Select **Open Browser** from the **File** menu or toolbar.

The Object Browser window is divided into three panes – Packages, Objects, and Members.

Packages pane

The Packages pane lists all of the packages from all of the file systems in the Repository. The first icon on the top of the pane is the **Show as tree** icon, which gives you the choice whether to view the packages as a tree (when selected) or a list (when deselected). The second icon is for expanding the whole tree. A pull-down menu in the upper left corner allows you to choose different package filters.

Objects pane

The Objects pane displays objects of the package selected in the Package pane. Three types of objects (classes, interfaces, and files without source) can be filtered. The seven toggle icons in the top of the pane represent filters for these objects. The first three icons (Class, Interface, and Class without source) represent general filters, meaning you can choose whether to display any objects of these types. The second group of icons (Public, Package, Protected, and Private) are filters for the class or interface according to their access modifiers.

For example, if only the Class, Private, and Package filters are selected, the following will be displayed:

- All classes with private and package-private modifiers
- All other non-class and non-interface objects

No classes with public or protected access and no interfaces at all will be shown.

By default, all seven filters are selected, meaning that all objects are shown, including classes and interfaces with any access modifier.

Whereas the Explorer uses a tree structure, the Objects pane uses a list structure. Whereas the Explorer tree would should show class `Innerclass` as a subnode of `Outerclass`, the Object Browser would display them in a list as the separate items `Outerclass` and `Outerclass.Innerclass`.

Members pane

The Members pane displays members of the object selected in the Object pane.

The first icon in the Members pane is a toggle switch that allows you to view members as bean properties and events. When this icon is selected, the other filters in the pane are disabled.

The next three icons represent filters for general member types (methods, variables, and constructors). As with class and interface objects in the Objects pane, you can further filter method, variable, and constructor members according to their access modifiers using the second group of icons (identical to the last four icons in the Objects pane).

Clicking on the last icon in this pane allows you to sort the members alphabetically within member type.

Tip: You can use the tool tip feature to find out the names of the filters and more information about the items listed in the panes. To get an item's tool tip, place the mouse cursor over the item and wait about a second for the tool tip to appear.

Using the Object Browser

Much like with the Explorer, you can use the Object Browser as a base for many tasks in the development cycle. Context menus are available for every item listed in the Object Browser, allowing you to do the following:

- Open objects or their members in the Editor window (and, if applicable, the Component Inspector and Form Editor window)
- Open HTML files in the Web Browser

- Cut, copy, and paste objects and their members
- Delete, rename and add packages, objects, and members
- Add new packages, objects, and members
- Compile or build an object or package
- Open up a separate Explorer window on a package

You can also double-click an object or member to open it up in the Editing workspace.

When you open a source file from the Object Browser, the Editor window opens up just below the Object Browser by default allowing you to easily switch between these windows. If multiple source files are open at the same time, each file has its own tab on the bottom of the Editor window, allowing you to go back and forth between files.

Note: The Object Browser is not specifically designed for designing visual applications. When working on visual projects, you may find it easier to work in the Editing workspace with the Explorer where, for example, you can easily edit properties for visual forms and double-click on a visual source node to open the Form Editor on that node. See "Developing Visual Classes" on page 84 for more information.

Creating package filters

If you have a lot of packages in your Repository, you may want to create custom package filters in the Package pane of the Object Browser to make it easier to view the objects you want.

To create a new package filter:

1. Make sure that the **Show as tree** icon is *not* selected.
2. Click the **...** button in the upper right corner of the Packages pane to bring up the Package Filter dialog.
3. In the Package Filter dialog, click **New**. A filter called `New Filter` will be automatically created.
4. Now a Change Filter Name dialog will appear. Enter a name for the new filter and click **OK**.

Next you will need to enter the specifications for your filter. In the details panel of the Package Filter dialog is a list of filter expressions. By default, the expression `com.*` is created for all new filters.

There are two ways to create a filter:

- Using the **wildcard** option, you can enter prefix (e.g. `com.netbeans*` suffix (e.g. `*debugger`), or infix (e.g. `*debugger*`) expressions to narrow down the number displayed packages.

- Using the **regular expression** option, you can create more complex filters using regular expressions (in standard POSIX format).

To add an expression to a filter:

1. Select the filter and click **Add**.
2. Click the radio button for **wildcard** or **regular expression** and type the expression in that field.
3. Click **Change**.

To change an expression in a filter:

1. Select the filter in the list of filters in the Package Filter dialog and select the expression to be changed in the Details list (e.g. `com.*`).
2. Continue with steps 2 and 3 of the add filter procedure.

Other buttons

- Pressing **Remove** takes the selected filter expression out of the selected filter.
- Pressing **Delete** removes the entire filter from the pull-down menu.

Browsing and exploring objects and their elements

You can view members of classes in both the Object Browser and the Explorer. Source files in a file system are represented as Java objects. Each object contains at least one class within the source as well as code representing members such as methods, constructors, and variables. In the Object Browser, the top-level and inner classes are shown in the Objects pane with all of their members appearing in the Members pane. In the Explorer, classes, named inner classes, and members are all represented hierarchically in subtrees of the Java objects.

Table 1: Source and form file icons

Icon	*Description*
	Java source file (both JavaBeans and non-JavaBeans files)
	Java file without source code (both JavaBeans and non-JavaBeans files)
	Runnable Java object (i.e. JavaBeans file or non-JavaBeans file that has a `main` method)
	Invalid Java source file (cannot be parsed)
	Form object
	Runnable form object
	Incorrect form object (cannot be parsed)

Invalid package declarations

Source files that have the wrong package named in their code are marked with `Invalid package declaration` in the tool tip. To rectify this, you can change the package name in the file or mount a different directory. As a shortcut, you can use the Open File feature to open mount a new directory and open the file in that directory. See "Adding a file to the IDE" on page 40.

Elements of Java objects

Subtrees of each Java object represent the hierarchy of elements. Each Java object contains at least one class which in turn contains elements such as constructors, methods, variables and inner classes. You can set behavior properties for each of these elements.

Table 2: Java Object Elements

Icon	*Elements*	*Properties*
▦	Class, Inner class	Name, modifiers, extended class, implemented interfaces
⊸	Interface, Inner interface	Name, modifiers, extended interfaces
●	Constructor	Name, modifiers, arguments, exceptions
●	Method (non-static)	like constructor, plus return type
■	Method (static)	like constructor, plus return type
●	Variable (non-static)	Name, modifiers, type, initial value
■	Variable (static)	Name, modifiers, type, initial value
●	Initializer (non-static)	
■	Initializer (static)	

Member accessibility

The elements listed above can have several kinds of accessibility. The icons of the elements consist of icons in the previous table and their accessibility flag(s).

Table 3: Icons for accessibility

Icon	*Access*
🔒	private
🔒	package private
🔑	protected
[no icon]	public

To create a new element

1 Select the class (or interface) in which you want to create the element.

2 Select **New** from the popup menu.

3 In the expanded submenu, choose the element you want to create.

Source synchronization

Source synchronization is a feature that helps to keep your Java source files current by:

- automatically generating an interface's methods for you when you implement that interface in a source file; and
- automatically updating all source files in the Repository implementing the interface when you change a method in or add a method to that interface.

Source synchronization works with all interfaces stored in the Repository and all standard interfaces in the Java 2 SDK. You can choose to synchronize source automatically just after parsing (which, by default, occurs after a two-second break in typing and caret movement) with or without confirmation, or explicitly by selecting **Synchronize** from the context menu of the implementing class in the Explorer.

All source files have a `Synchronization Mode` property on their property sheet (accessed right-clicking on the node and selecting **Properties** or clicking the **Toggle Property Sheet** icon in the Explorer when the node is selected). The property can be set to:

- **Do not synchronize**.

- **Confirm changes** – the default setting. The Confirm Changes dialog appears which allows you to specify which methods are to be synchronized.
- **Without confirmation** – all interface methods are automatically generated for the implementing classes without a dialog, prompting you to confirm the changes.

Source Synchronization property sheet

You can adjust general source synchronization settings on the property sheet for the `Project Settings / Java Sources / Source Synchronization` node in the Explorer. The properties are:

- **After Parsing Without Errors** – If `True`, synchronizes after parsing without errors.
- **After Save Only** – If `True`, synchronizes after saving.
- **Return Generation Mode** – Generates either nothing, an exception, or the string "return null" when creating a new method declared to return a value.
- **Synchronization Enabled** – If `False`, all synchronization is turned off.

Synchronizing source

To synchronize (with confirmation) when implementing a new interface in a class:

1. In the Editor window, type `implements` *`InterfaceName`* in the class declaration line of the source file.
2. Wait a few seconds for the source to be parsed, or save or compile the file. The Confirm Changes dialog will then appear listing all of the methods to be added to the class.
   a. Select **Process All** to add all of the interface's methods to the class; or
   b. Select any combination of the methods in the **Changes List** and select **Process** to add only those methods to the class; or
   c. Select **Close** to prevent any of the methods from being added to the class.

The Confirm Changes dialog also lets you change the synchronization mode for the class.

To synchronize (with confirmation) when changing or adding methods to an interface:

1. Add the method(s) to the interface code and then wait for parsing to occur and the Confirm Changes dialog to appear. The dialog will display the updated methods for the first class found that implements the interface.
2. Follow step 2 from the previous procedure.

If you have multiple source files in the Repository that implement the changed interface, a Confirm Changes dialog will appear for each one of them.

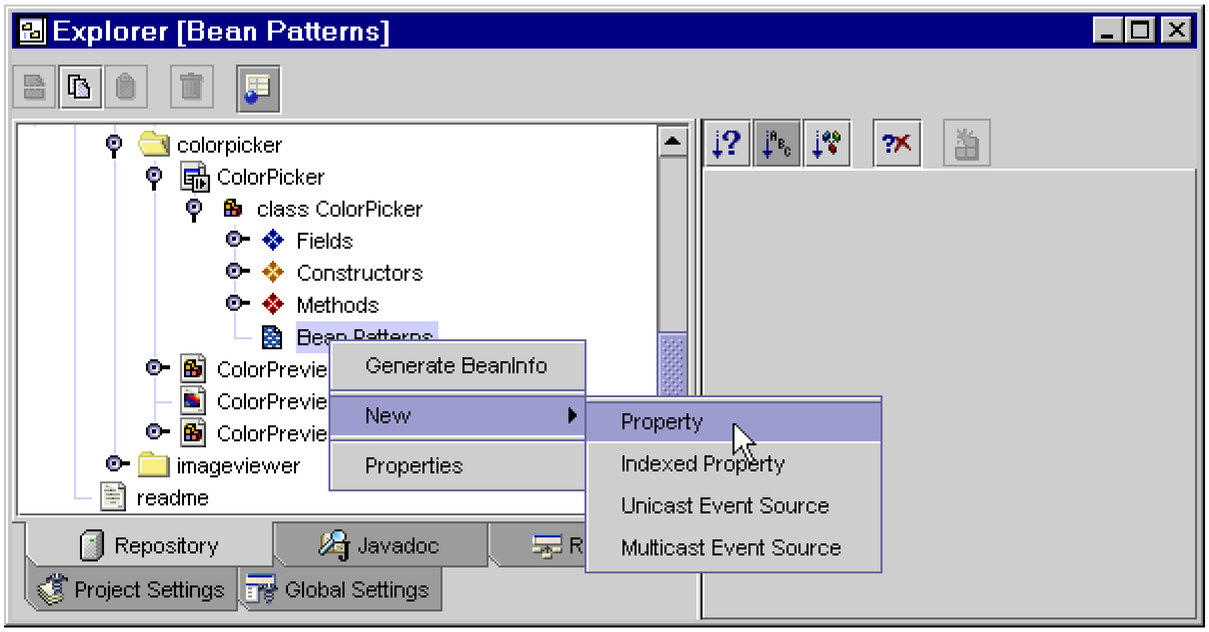
If you have automatic synchronization disabled on a specific source file or for all source files, you can still synchronize a class with its interface's methods by selecting **Synchronize** from the class's context menu.

Note: The **Synchronize** command only works when selected on class objects (not interfaces).

Developing JavaBeans™ components

Forte for Java Community Edition 1.0 provides tools to make generation and customization of JavaBeans™ Components faster and easier. You can create the standard parts of your bean – such as properties, listener support, default constructor, and BeanInfo – without having to write any code manually.

In Forte for Java, creation of JavaBeans components begins with any class that has source code. In the Explorer, you will find `Bean Patterns` as a subnode of any such class. All of the operations for maintenance of property and event sets – thus all of the things that determine the characteristics of the bean – can be accessed and modified from this subnode.



Note: `Bean Patterns` also includes non-public property and event mechanisms, which are not recognized as JavaBeans patterns in introspection. This is because `Bean Patterns` can be used to set property and event mechanisms in standard classes which are not true JavaBeans components.

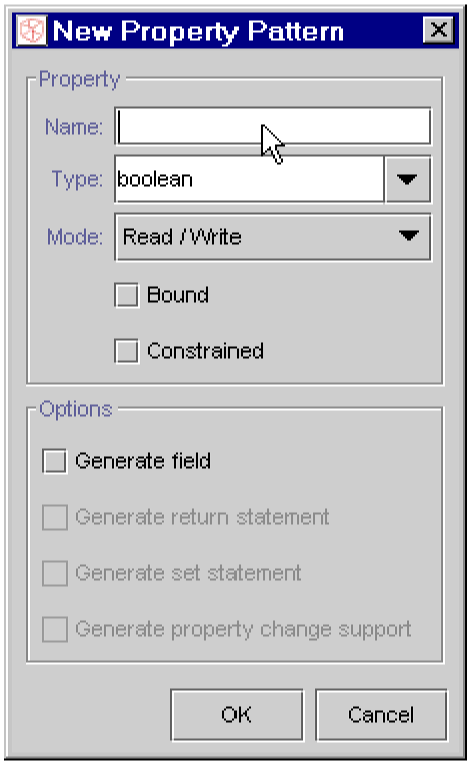
For a more thorough review of JavaBeans Components and concepts such as introspection, see http://java.sun.com/beans/.

Creating properties

To create a property in a class source:

1 Find the class in the Explorer, expand its node, and right-click on `Bean Patterns`.

2 In the popup menu, select **New | Property**. The New Property Pattern dialog will appear, which will allow you to customize the code to be generated for the property.



3Type in a name for the property. (The name must be a valid Java identifier.)

4In the **Type** combo box, select the type of property from the list, or type in a class identifier.

5In the **Mode** combo box, choose whether to have the getter method (`READ ONLY`), setter method (`WRITE ONLY`), or both methods (`READ/WRITE`) generated.

6Check the **Bound** (meaning that property change events must be fired when the property changes) and/or **Constrained** (meaning that the property change can be vetoed) options, if applicable to the property. The usefulness of checking these options is enhanced if you also check the **Generate Property Change Support** option – see below.)

If you click **OK** after these steps, the definitions of the methods will be generated. However, you can also check all or any combination of the following options:

- If you check the **Generate field** option, a private field is generated. This field will have the same name and type as the property.
- If you check the **Generate return statement**, a statement that returns the field (e.g. `return myProperty;`) is inserted in the body of the getter method.
- If you check the **Generate set statement** option, a statement setting the value of the property field to the value of the setter parameter will be inserted into the body of the setter method.
- If you check **Generate property change support**, all of the code needed for firing `PropertyChangeEvents` (for bound properties) and `VetoableChangeEvents` (for constrained properties) will be generated in the bodies of the setter methods. In addition, code to declare and initialize the property change support object is generated.

Creating indexed properties

You can also create indexed properties. Indexed properties are usually implemented as arrays or vectors and allow you to set and get values using an index. Indexed getter and setter methods have a parameter which specifies the index of the element to be read and written.

To create an indexed property in a class source:

1 Find the class in the Explorer, expand its node, and right-click on `Bean Patterns`.

2 In the popup menu, select **New | Indexed Property**. The New Indexed Property Pattern dialog will appear, which will allow you to customize the code to be generated for the property.

3 Proceed according to steps 3 through 6 above in the instructions for creating a simple (non-indexed) property.

4 Check the items in the Options section appropriate for the property. These four options are analogous to the options for a simple property.

In addition, the indexed properties may have getter and setter methods which enable reading and writing all elements (the whole array). The checkboxes in the Non-Index Options panel enable you to add a getter with or without a `return` statement and a setter with or without a `set` statement.

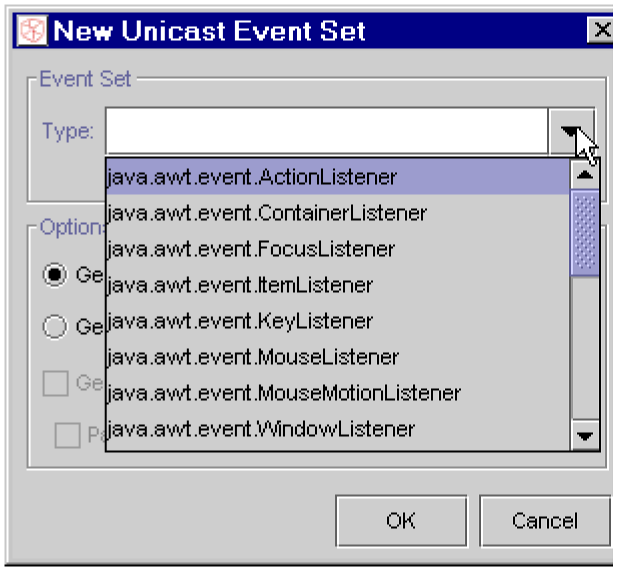
5 Click **OK**.

Creating event sets

You can also add event sets to your bean.

To add an event set deliverable to only one listener:

1 Right-click on the `Bean Patterns` subnode of your class and select **New | Unicast Event Source** from the context menu. The New Unicast Event Set dialog will appear on the screen.



java.awt.event.ActionListener
java.awt.event.ContainerListener
java.awt.event.FocusListener
java.awt.event.ItemListener
java.awt.event.KeyListener
java.awt.event.MouseListener
java.awt.event.MouseMotionListener
java.awt.event.WindowListener

2Use the drop-down list in the **Type** field to specify any listener interface (event class type) that extends `java.util.EventListener`.

3Click one of the two radio buttons below the **Type** field to select how you want the event set implemented. The choices are:

•**Generate empty** – generates an empty implementation

•**Generate implementation** – generates a simple implementation for one listener

4Check the **Generate event firing methods** box if you want to generate a method corresponding to every method in the listener interface to fire the event to all listeners.

5 To specify how the event will be transferred to this method, check the option **Pass event as parameter**. This adds parameters to the firing events. The type of parameter (`EventObject` subclass) will be the same as the type of parameter of the corresponding method in the listener interface.

If you leave the **Pass event as parameter** option unchecked, the firing method will have the same parameters as the constructor of the event object class (subclass of `EventObject`), and this constructor will be called in the body of the firing method. If there are multiple constructors for the event class, the generator behaves as if the **Pass event as parameter** option is checked.

When you click **OK**, an `add`*`EventName`*`Listener` method and a `remove`*`EventName`*`Listener` method, along with firing methods if you specified them, will be added to your source.

For multicast event sources you can specify how to implement the adding of listeners and firing of events.

To add an event set deliverable to more than one listener:

1 Right-click on the `Bean Patterns` subnode of your class and select **New | Multicast Event Source** from the context menu. The New Multicast Event Set dialog will appear on the screen.

2 In the **Type** field, specify any listener interface (event class type) that extends `java.util.EventListener`.

3 Click one of the three radio buttons below the **Type** field to select how you want the event set implemented. The choices are:

- Generating an empty implementation
- Using a simple `ArrayList` implementation

- Using the `EventListenerList` support class from the `javax.swing.event` package

4 Follow steps 4 and 5 in the procedure for adding unicast events.

Generating Bean Info

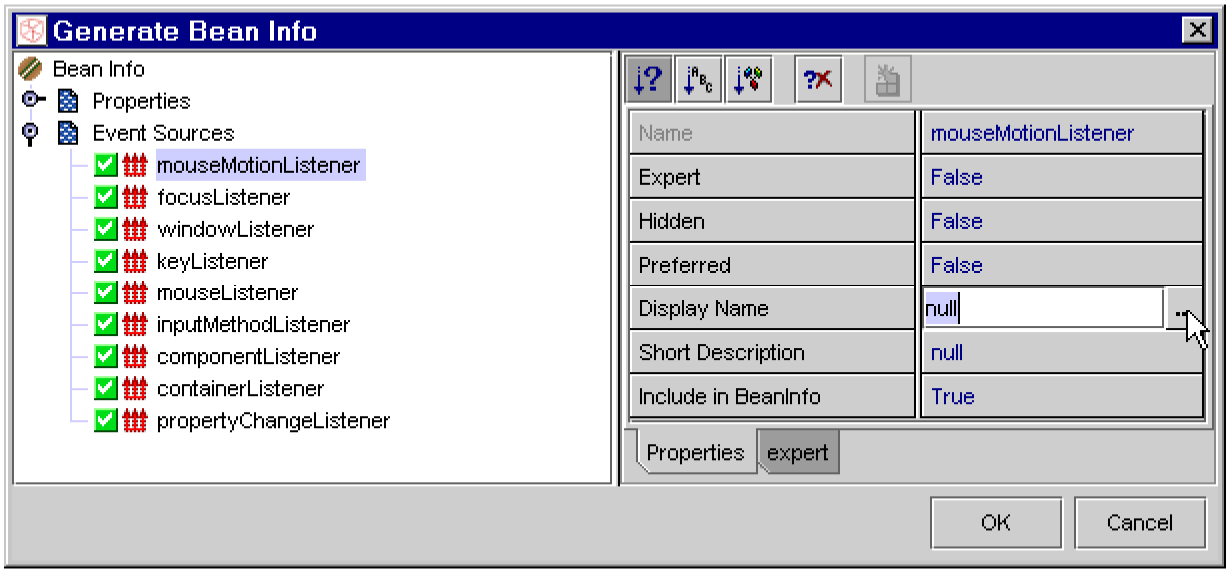
BeanInfo is a class which contains information about the JavaBean, such as properties, event sets, and methods. BeanInfo is an optional class, and you can choose what information will be included in it. In BeanInfo you can also specify additional properties of the JavaBeans properties and BeanInfo, including icons, display names, etc.

By default, all superclass beaninfo, public properties and event sets are included in `BeanInfo` (permitting customization of various attributes), and all public methods are discovered by introspection.

To generate BeanInfo for a class:

1 Right-click on the `Bean Patterns` node in the class and select **Generate BeanInfo** from the popup menu.

The Generate Bean Info dialog, which looks much like the Explorer with its Property Sheet pane open, will appear. The left panel shows the three main nodes (`Bean Info`, `Properties`, and `Event Sources`) and the right panel shows the properties of the selected node.



Name	mouseMotionListener
Expert	False
Hidden	False
Preferred	False
Display Name	null
Short Description	null
Include in BeanInfo	True

2 Make any desired changes to the properties and then click **OK**.

Note: The JavaBeans Components properties referred to in the `Properties` node should not be confused with the word “properties” as generically used in Forte for Java to refer to characteristics and settings (as displayed in each object’s property sheet) of parts of the IDE and objects stored in it. Thus, in the Bean Info Generation dialog, we refer to properties of JavaBeans properties.

Bean Info node

The first four properties (e.g. `Icon 16x16 Color`) allow you to designate icons for the bean by entering the relative path to and name of a graphic file.

The next two, `Default Property Index` and `Default Event Index`, apply to the whole BeanInfo class. In these two properties, you can specify a default property or event that may be declared in the BeanInfo to be customizable. The values are the indexes of the default property and default event set in the `PropertyDescriptor` and `EventSetDescriptor` arrays (respectively) and are set to -1 if there is no default property.

For more information on these and the other properties, see *BeanInfo* in the API documentation that comes with the Java™ 2 SDK or check Sun's website, http://java.sun.com/products/jdk/1.2/docs/api/java/beans/BeanInfo.html

Properties and Event Sources nodes

These nodes have only one property, `Get From Introspection`. By default, this property is set to `False`. If you set the value to `True`, then (when the bean is used) all information about the properties (or event sets) will be taken from introspection. Thus the generated `BeanInfo` will return a null value instead of an array of descriptors of properties (or event sets).

This feature is particularly useful if you want to customize either JavaBeans properties or event sources, but not both. Thus you can customize one (properties or event sources) and have the information about the other taken from introspection.

If you right-click on either of these nodes, you can choose from the popup menu to either include or exclude all of the JavaBeans properties or event sets from the `BeanInfo`.

Subnodes of the Properties and Event Sources nodes

The subnodes are the JavaBeans properties and event sources themselves. Each of these subnodes has two groups of properties. The first group, under the **Properties** tab, are general properties and apply to both JavaBeans properties and event sets. The **Expert** tab holds settings specific to properties or event sets.

The most important property here is the `Include in BeanInfo` property, which is set to `True` by default. If you set this property to `False`, the JavaBeans property or event set will not be shown in the BeanInfo, either to users or to other classes. You can also change this value by right-clicking on the property or event source and selecting **Toggle Include**. When `True`, the icon has a green square with a white check mark; when `False`, it displays a red square with a white X.

In the **Expert** tab, you can change the mode of a JavaBeans property to make it more restrictive (i.e. you can change a `READ/WRITE` to `READ ONLY` or `WRITE ONLY`).

Note: If you have any JavaBeans properties with non-standard names, source for these is not automatically generated in the `BeanInfo`. You can enter such code manually in the Editor.

In addition, if an indexed JavaBean property has a non-indexed getter and setter, you can specify whether these methods are specified in the `BeanInfo`.

Tip: If you want to set a property sheet property to the same value for multiple JavaBeans properties or event sets, you can select several nodes at once and set the properties for all of them.

Editing BeanInfo source

After you have generated `BeanInfo` from the Generate Bean Info dialog, you can manually edit the `BeanInfo` in the Editor window (except for the blue guarded blocks).

Regenerating BeanInfo

You may regenerate `BeanInfo` for classes that already have `BeanInfo`. If there is already a `BeanInfo` class for a bean, any changes you make in the Generate Bean Info dialog or in the `BeanInfo` source code will be saved.

Customizing JavaBeans components

When writing in Java, it may be useful for you to take existing beans and alter them to better suit your purpose. The **Customize Bean** command allows you to create an instance of a JavaBean, customize its properties, and save it as a serialized prototype. This may be done on a Java class or on an already serialized prototype (`.ser` file). The class must be public and have a public default constructor, and it must also be serializable to make a serialized prototype from the customized settings.

To customize a JavaBean

1 Right-click on the bean in the Explorer and select **Customize Bean** from the context menu.

   The property sheet for the JavaBean will open up. If the bean is a visual component (meaning that it extends `java.awt.Component`), the component itself will also appear on the screen.

2 Make the desired changes in the property sheet.

3 Click **Serialize** or **Serialize as** to create a serialized prototype (`.ser` file) of the bean with its customized properties.

Serialize As allows you to select a name and location for the `.ser` file. **Serialize** (available for `.ser` files only) saves the bean into the same `.ser` file it originated from, overwriting the previous content.

If you do not want to keep the customized settings, press the **Cancel** button to close the customization window.

Adding and modifying service types

Forte for Java Community Edition 1.0 provides editable **service types** to give you more control over how your applications are compiled, executed, and debugged. These service types specify not only which compiler, executor, or debugger to use but also how the service is to be invoked (what the path to Java is, what arguments are used, etc.).

The IDE comes with a set of default service types which suit most development tasks. A compiler type, executor, and debugger type is assigned to each class by default. You can set service types for each class individually in the Repository.

You can also modify the existing service types and create new ones. For example, the default service type for external execution of applications may not suit every one of your classes, so you may want to have multiple execution service types (or "executors"). You can create a new executor type for external execution, and use this executor for some of your classes while you use the default executor for other classes.

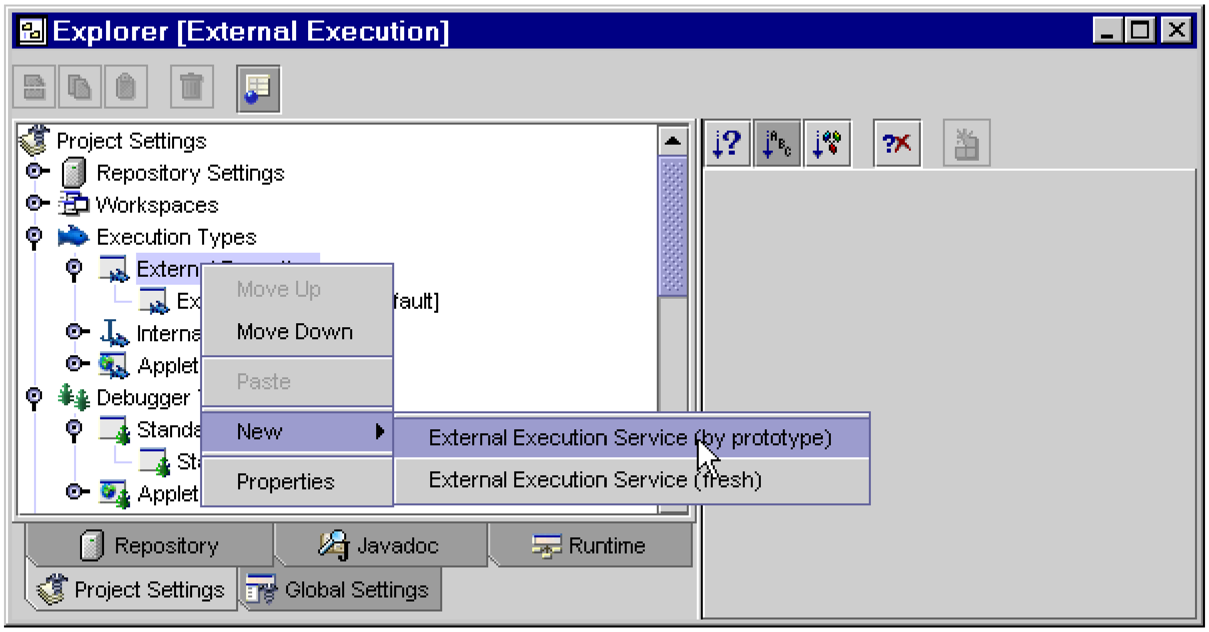
Service types are represented as subnodes under the various categories of `Compiler Types`, `Debugger Types`, and `Executor Types` under the **Project Settings** tab in the Explorer. For all categories of compilers (e.g. external or internal), executors (external, internal, applet, etc.) and debuggers in Forte For Java Community Edition 1.0, the IDE comes with one service type.

Adding service types

To add a new service type:

1 Under the **Project Settings** tab in the Explorer, expand the `Compiler Types`, `Executor`

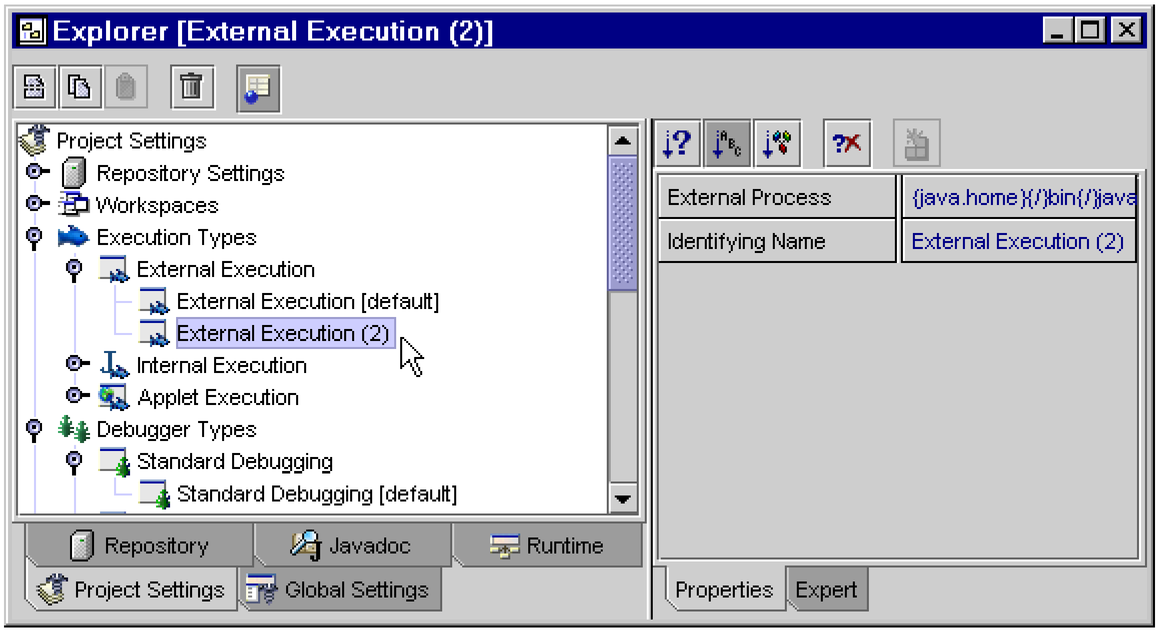
`Types`, or `Debugger Types` node, depending on the kind of service type you want to add.



2 Right-click on the node for the category of service type you would like to add and select **New | *[service type]* Service (by prototype)** or **New | *[service type]* Service (fresh)** from the context menu.

The first option creates a copy of the default service type (with its current configuration) for that category. The second option creates a fresh service type with a generic configuration

3 Expand the category's node. You will see at least two subnodes. The last subnode is the newly created service type.



External Process	{java.home}{/}bin{/}java
Identifying Name	External Execution (2)

4 Select the newly added node and, on its property sheet, modify the `Identifying Name` property to give the new service type a distinct name (something that you will recognize when

using the drop-down list to choose a service type for an individual class). You can type the name directly into the field and press ENTER, or you can click on the property's value and then click on the **...** button that appears to use the custom property editor.

Tip: If you want to make a newly added service type the default for its category (meaning that subsequent new objects, unless configured to do otherwise, will use that service type), you can right-click on the service type and select **Make Default** from the context menu.

Configuring service types

After you have added a new service type, you can then configure it. (You can also configure already-existing service types).

To configure a service type:

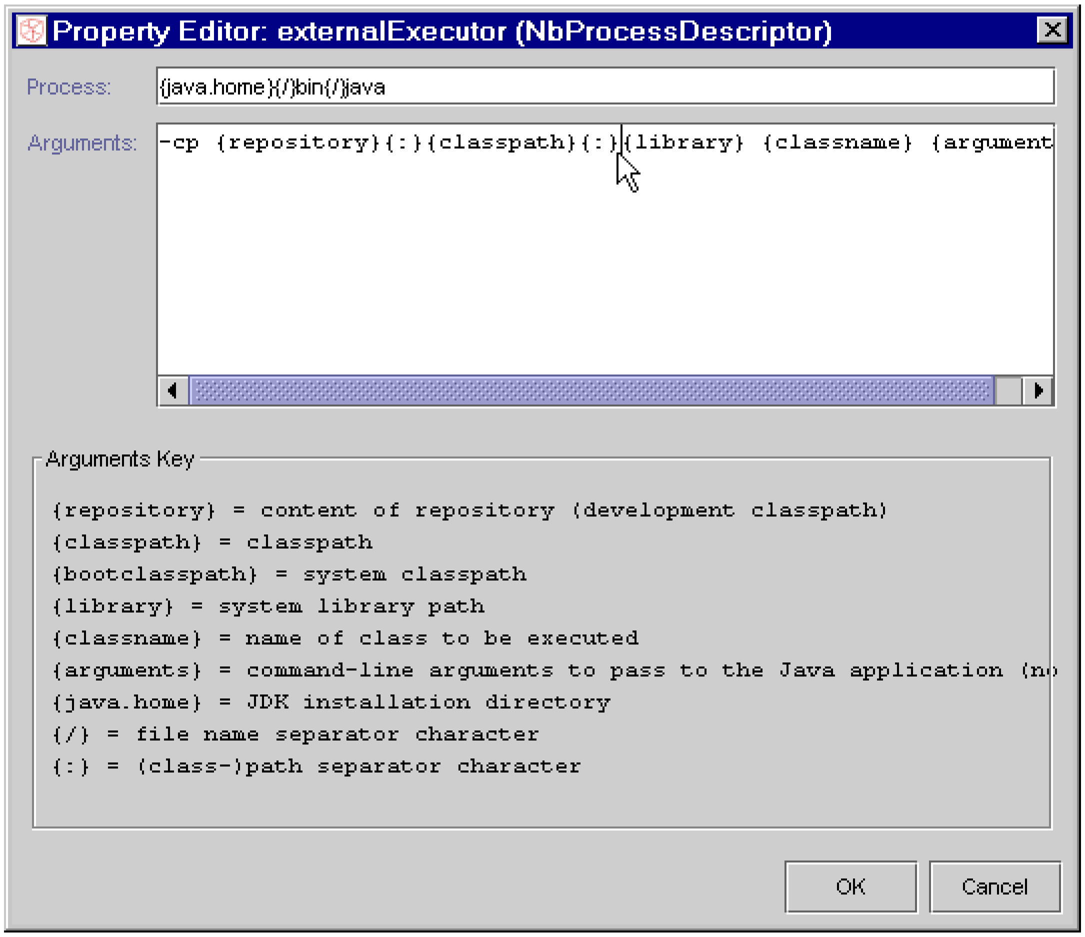
1 Select the service type's node (under the service type category under `Compiler Types`, `Executor Types`, or `Debugger Types` under the **Project Settings** tab in the Explorer) and open its property sheet.

2 Modify any of the editable properties under the standard **Properties** or the **Expert** tabs.

Note: Classes have no way have registering the name changes of the service types they are using. Therefore, if you change the `Identifying Name` property of a service type already being used by any classes, those classes will then revert to the current default service type for that category (which is only a problem if you are changing the name of a non-default service type).

Process Descriptor property editor

Some of the service types have a property (e.g. `External Executor` for external executors and `External Compiler` for external compiler types) that can be edited with the Process Descriptor property editor. In this custom property editor, you can set the process and arguments for the service type. The Arguments Key panel displays the substitution codes (enclosed in curly braces) which you can use (and are used by default) in defining the service type.



```
{java.home}{/}bin{/}java
```

```
-cp {repository}{:}{classpath}{:}{library} {classname} {argument
```

```
{repository} = content of repository (development classpath)
{classpath} = classpath
{bootclasspath} = system classpath
{library} = system library path
{classname} = name of class to be executed
{arguments} = command-line arguments to pass to the Java application (no
{java.home} = JDK installation directory
{/} = file name separator character
{:} = (class-)path separator character
```

Note: The `{arguments}` substitution code uses any arguments entered in the `Arguments` property of the class being run.

Tip: If you use a lot of different service types, you can save time by incorporating them into object templates, which you then can use each time you create a new object. Just create a generic class, set the desired service types on the class's property sheet, and then save the class as a template. For more information, see "Creating new objects from templates" on page 148.

Editing service types from the Repository

You can also access the property sheet for the different service types from the property sheet of an individual class's Repository node.

To access a service type's property sheet from the Repository

1 Select a class under the **Repository** tab in the Explorer and open its property sheet.

2 Under the **Execution** tab of the property sheet, select the `Compiler` (or `Executor` or `Debugger`) property and then press the **...** button to invoke the custom property editor.

3 In the list on the left side of the custom property editor, select the compiler/executor/debugger you want to modify. The property sheet for that service type will appear. (You can select the `External Compiler` (or `External Executor` or `Debugger`) property and press the **...** button in order to edit the service type's process and arguments.)

Note: When you reconfigure a service type, even if you are invoking the custom property editor(s) from the property sheet of a class, the changes you make to the service type will affect all other classes that use that service type.

Removing service types

To remove a service type:

◊ Right-click on the service type's node in the Explorer and select **Delete** from the context menu.

Note: It is not possible to delete the default service type for a given category.

Searching Javadoc

Forte for Java Community Edition 1.0 comes with a module that allows you to easily browse standard API documentation and other Javadoc files from within the IDE.

Preparing the Javadoc Repository

Before you can do Javadoc searches within the IDE, you need to add directories (or JAR or ZIP files) with standard API documentation to the search path.

There should be two directories already mounted in the Javadoc Repository (the **Javadoc** tab in the Explorer):

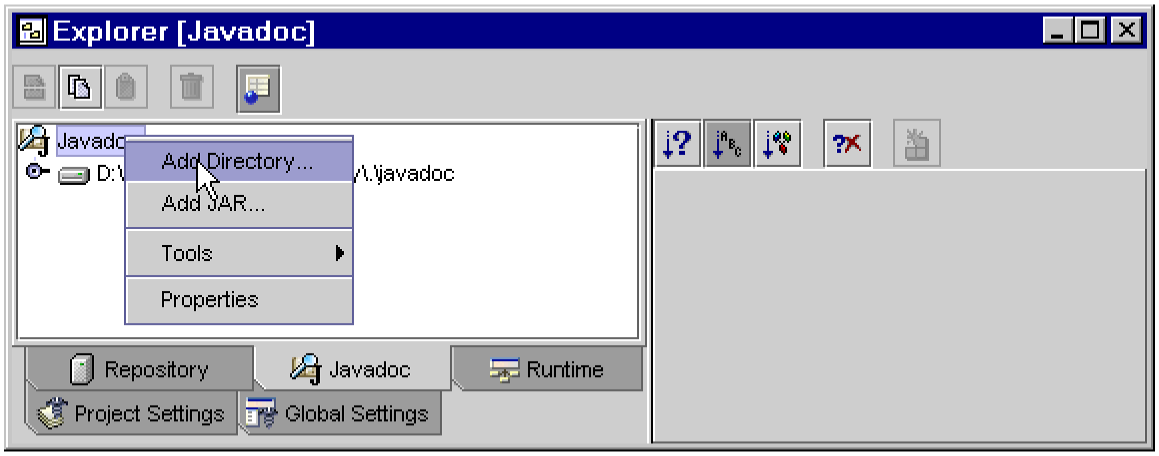
- the Java API `docs` directory (if it is installed under the Java™ 2 installation directory). It will appear there with a name such as `/usr/jdk1.2.2/docs/api`.
- the `Javadoc` directory (which can be found in the root of the IDE's installation directory).

Note: By default, the Java API `docs` directory is set to hidden, meaning it will not be displayed in the main Repository (under the **Repository** tab). If you would like to have it displayed in the main Repository (in addition to the Javadoc Repository), go to the **Project Settings** tab in the Explorer, expand the `Project Settings / Repository Settings` node, select the Java API directory node and set its `Hidden` property to `False`.

If neither directory is present or you would like to add a new directory with Javadoc documentation, you can add them by following this procedure.

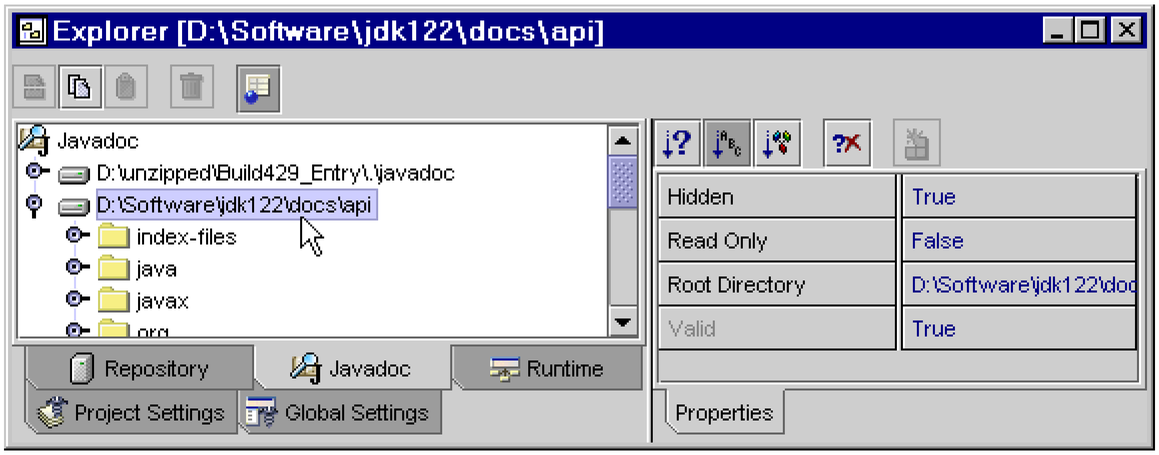
To add a Javadoc documentation directory to the Javadoc Repository:

1 In the Explorer, select the **Javadoc** tab, right-click on the `Javadoc` node, and choose **Add Directory...** from the popup menu. (If you are adding a JAR or ZIP file with Javadoc documentation, choose **Add JAR...** for JAR or ZIP files).



2 In the Mount Directory dialog (or Mount JAR Archive dialog when mounting a JAR or ZIP archive) that appears, choose the directory with the Javadoc documentation.

When selecting the Javadoc documentation directory, you should look at all original documentation distributed with the SDK and be sure that it has an `index-files` directory or `index-all.html` file in the top level or the second level of its hierarchy.


Explorer [D:\Software\jdk122\docs\api]
Javadoc
D:\unzipped\Build429_Entry\.\javadoc
D:\Software\jdk122\docs\api
index-files
java
javax
org
Repository
Javadoc
Runtime
Project Settings
Global Settings
Properties

Hidden	True
Read Only	False
Root Directory	D:\Software\jdk122\doc
Valid	True

Searching in Javadoc directories

Once the Javadoc directory is installed, you can begin searching Javadoc documentation from within the IDE's JavaDoc module.

To search on Javadoc documentation, either:

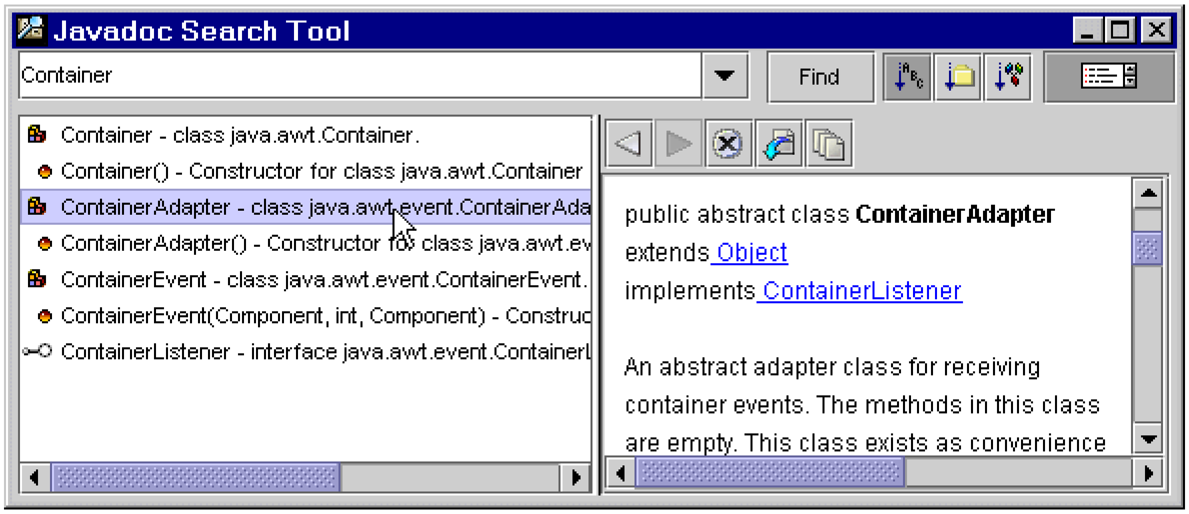
1 Invoke the JavaDoc Search Tool dialog by pressing CTRL+F1 or selecting **Help | JavaDoc Index Search** from the Main Window.

2 Use the combo box in the dialog to type or select the search string, and then press **Find** or press ENTER on your keyboard.

or:

◊ Select the string or click on the word you want to search in the Editor window, and then press CTRL+F1 or select **JavaDoc Index Search** from the **Help** menu.

The `JavaDoc Search Tool` will then begin searching for your string.



Container - class java.awt.Container.
Container() - Constructor for class java.awt.Container
ContainerAdapter - class java.awt.event.ContainerAda
ContainerAdapter() - Constructor for class java.awt.ev
ContainerEvent - class java.awt.event.ContainerEvent.
ContainerEvent(Component, int, Component) - Construc
ContainerListener - interface java.awt.event.ContainerL

public abstract class **ContainerAdapter**
extends Object
implements ContainerListener

An abstract adapter class for receiving container events. The methods in this class are empty. This class exists as convenience

Search dialog

The top part of the JavaDoc Search Tool dialog consists of a combo box for selecting or typing the search string, the **Find** button for starting the search (after the search is started, it changes to **Stop**).

The three icons to the right of the **Find** button serve as toggles for sorting the pages found. Selecting the first sorts the items alphabetically, selecting the second sorts by package and alphabetically, and the third sorts by element (exceptions, classes, constructors, etc.).

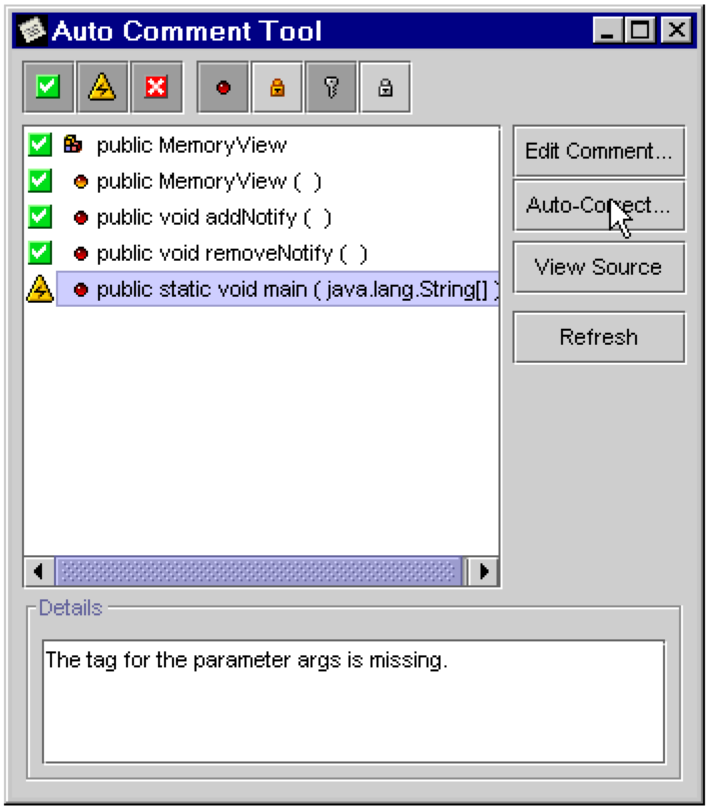
Below the combo box and icons are two panes. The first pane displays pages found in the search, and the second pane shows the page selected in the first pane. The left pane can be expanded to cover the whole dialog area by deselecting the icon to the right of the sort icons. The size of the panes can be changed by dragging the divider in the center.

Using Javadoc Auto Comment

The Auto Comment utility enables you to comment your source code automatically and view all parts of your source code (methods, constructors, inner classes, variables etc.) and document them individually.

Auto Comment Tool dialog

You can open the Auto Comment Tool dialog by right-clicking on a source file or one of its elements in the Explorer or Object Browser and choosing **Tools | Auto Comment...**. The AutoComment Tool dialog consists of two panes, seven filter icons and four buttons.



public MemoryView
public MemoryView ( )
public void addNotify ( )
public void removeNotify ( )
public static void main (java.lang.String[])

The tag for the parameter args is missing.

The first pane lists constructors, variables and other elements for comment. Below that, the `Details` pane shows information about missing or invalid comments on the item selected in the first pane.

The first three filter icons, respectively, enable you to view class elements

1. with completely valid comments

2. with partially valid comments (something could be missing or invalid in the Javadoc tags)
3. without comment

The next four icons are filters which enable you to view public, none, protected and private elements of class source.

The dialog also includes the following buttons:

- **Edit Comment...** – invokes the JavaDoc Comment dialog, where you can edit the comment. See the following section for more information.
- **Auto-Correct...** – opens Javadoc Comment window and automatically corrects any errors in the tags.
- **View Source** – opens the source file which you are currently documenting and moves the caret to the element selected in the JavaDoc Comment dialog.
- **Refresh** – refreshes the display in the first pane to reflect any changes made in the source file (e.g. through the Editor).

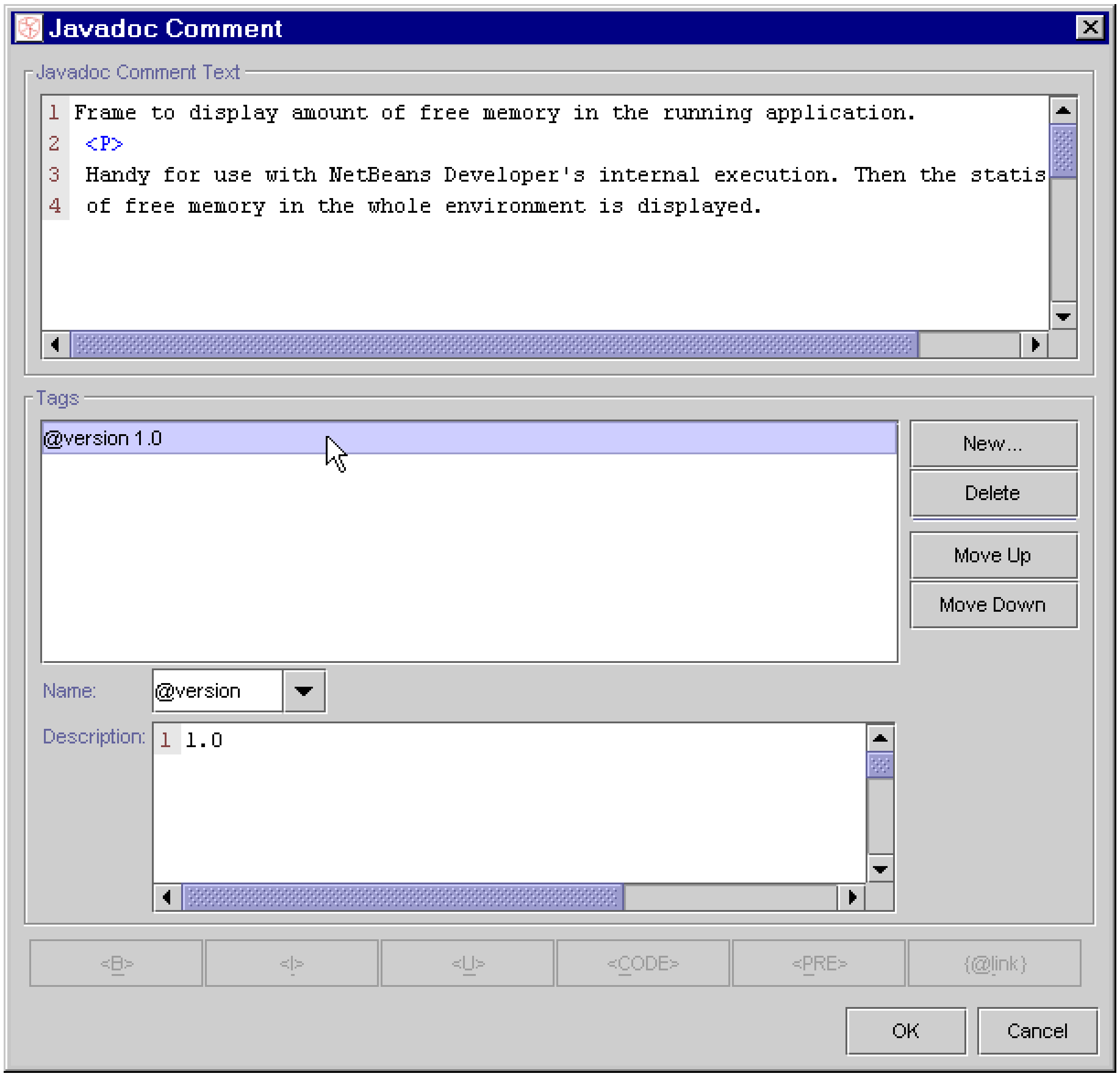
JavaDoc Comment dialog

The JavaDoc Comment dialog lets you add comments to class elements separately.

To open the JavaDoc Comment dialog:

◊ Press **Edit Comment** in the AutoComment Tool dialog; or

◊ Open the property sheet for the element you want to document, click on the value of the `JavaDoc Comment` property, and then click on the **...** button that appears.



```
Frame to display amount of free memory in the running application.
 <P>
 Handy for use with NetBeans Developer's internal execution. Then the statis
 of free memory in the whole environment is displayed.
```

@version 1.0

Name: @version

Description: 1.0

The JavaDoc Comment dialog has two main panes. The **JavaDoc Comment Text** pane displays comment text. The **Tags** pane displays all Javadoc tags used in the comment. When you select a Javadoc tag, a combo box appears along with the **Description** text field. (If the `@see` tag is selected, two read-only text areas named `Class` and `Type` also appear.)

In the Description pane, you can change the description of the Javadoc tag selected in the combo box. In the combo box you can change the selected Javadoc tag to another Javadoc tag.

There are four buttons on the right side of the **Tags** pane:

- **New** – invokes the New Tag dialog, which allows you to choose from predefined Javadoc tags or type another tag. It is element sensitive, so only appropriate tags on class members are shown.

- **Delete** – deletes the selected tag in the **Tags** combo box.
- **Move Up** and **Move Down** – change the order of the Javadoc tags in the comment by moving the selected tag up or down.

On the bottom of the JavaDoc Comment dialog are buttons with predefined HTML tags

and the Javadoc `{@link}` tag, which you can add to your comment and have displayed in the **JavaDoc Comment Text** pane.

Press **OK** to accept all changes or **Cancel** to reject all changes.

Note: For detailed information about Javadoc tags, visit Sun's Javadoc web page, at http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html#javadoctags.

Generating Javadoc documentation

You can have documentation for entire classes and packages generated automatically.

To generate Javadoc documentation:

1 Select the packages and/or source files for which you would like to create documentation in the Explorer (under the **Repository** tab) or in the Object Browser (**Packages** pane only).

2 Right-click on the object's node and select **Tools | Generate JavaDoc** from the context menu. A dialog appear asking you to name the directory where you want the Javadoc documentation generated.

By default, the generated Javadoc documentation will be stored in the directory `javadoc` in the IDE's home directory.

JavaDoc module properties

There are property sheets for both Internal Javadoc and Standard Doclet.

To view or change Javadoc properties:

1 Expand the `Project Settings / Documentation` node in the Explorer.

2 Select the `Internal Javadoc` or `Standard Doclet` node.

3 Select the **Toggle Property Sheet** icon or right-click on the node and select **Properties** from the context menu.

For more information on Javadoc properties, go to the Javadoc page on Sun's website: http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javadoc.html.

Changing the directory for generated Javadoc documentation

To change the directory for generated documentation:

1. Open the property sheet for **Project Settings | Documentation | Standard Doclet**.
2. Select the `Destination` property and change the directory manually or click the **...** button that appears to invoke the File Chooser and change it that way.

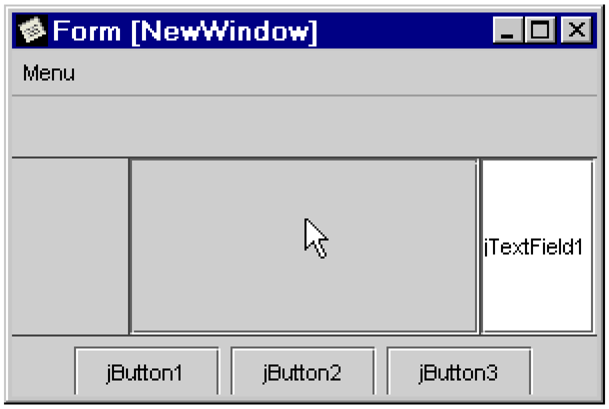
Chapter 5

Developing Visual Classes

This chapter details how to create visual forms in Java with Forte for Java Community Edition 1.0, using the Form Editor, templates, and other features that make the development process itself more visual and intuitive. You will also learn how to add new JavaBeans to the IDE for use in your development work.

Designing visually with the Form Editor

Perhaps the most significant feature that an IDE can provide for a developer is assistance in designing graphical interfaces for applications and applets. For this purpose, Forte for Java Community Edition 1.0 provides the **Form Editor**, which allows you to build your forms visually and have the code for them automatically generated.



The Form Editor, with its powerful open design based on JavaBeans architecture, allows you immediately to use any JavaBean for visual development. Because they use standard AWT and JFC components, generated forms do not depend on any proprietary code or components. No classes need to be bundled with your forms as the Form Editor generates code that is entirely independent of the IDE (unless you use Absolute Layout, a special Forte for Java feature).

Opening the Form Editor

To open a Form Editor window for existing forms, double-click on the form object in the Repository or select **Open** from its popup menu.

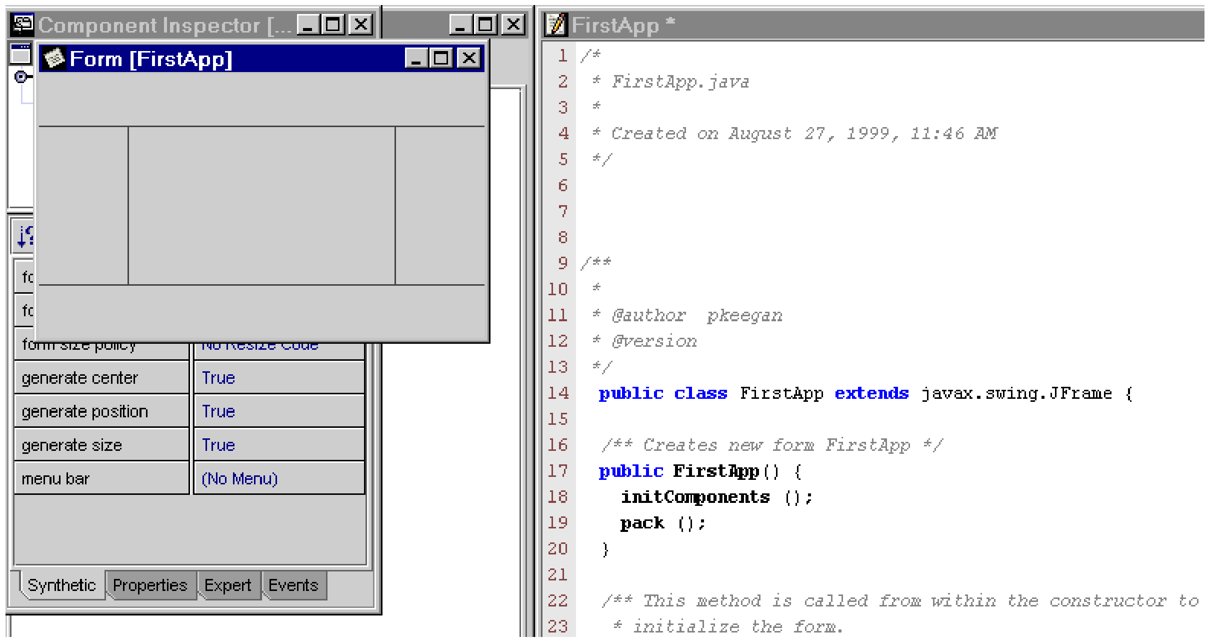
When a form is opened, three windows are displayed:

1. **Form Editor window** – the design-time view of the form.

2. **Editor window** – contains the Java source for the form. If the Editor is already open, the form is opened on a new tab within that Editor window.

3. **Component Inspector** – displays the hierarchy of components on the active form, including non-visual components such as the layout manager and menus. The current selection of components is highlighted. Tabs on the property sheet in the bottom panel display the synthetic, general, and layout properties as well as the events of the selected component. See "Property

Sheet pane in the Component Inspector” on page 90.



fo	
fo	
form size policy	No Resize Code
generate center	True
generate position	True
generate size	True
menu bar	(No Menu)

```
/*
 * FirstApp.java
 *
 * Created on August 27, 1999, 11:46 AM
 */



/**
 *
 * @author  pkeegan
 * @version
 */
 public class FirstApp extends javax.swing.JFrame {

  /** Creates new form FirstApp */
  public FirstApp() {
    initComponents ();
    pack ();
  }

  /** This method is called from within the constructor to
   * initialize the form.
```

Note: In Forte for Java Community Edition 1.0, forms are stored in XML format. When opening forms saved with NetBeans Developer 2.0, 2.1, X2 or early betas of 3.0, you will be asked whether you want to convert the form to new format. If you click **No**, the form editor will have the same features as in X2 and the saved form will be compatible with X2. However, some of the new features, such as multiple property editors per property and most synthetic properties on components, will not be available. If you click **Yes**, the form will be saved in the new format, and it will not be possible to open it in these versions of NetBeans Developer.

Creating a new form

Use the **New From Template** command to create a new form from template and open it in the Form Editor.

Forte for Java supports nine basic form types from both the Java Abstract Windowing Toolkit (AWT) and the newer Swing API. The table below lists them.

Table 4: Basic form types supported by Forte for Java

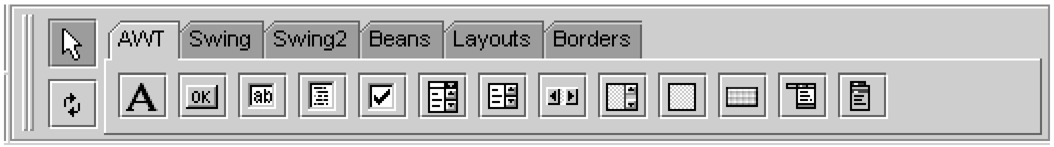
Name	***Description***
Frame	AWT Frame (top-level application window).
Dialog	AWT Dialog (modal or non-modal window for user feedback).
Applet	AWT Applet (embeddable program run by a Web browser or other applet viewer).

Name	***Description***
Panel	AWT Panel (container for holding parts of an interface—can in turn be used in any other container, such as a Frame, Panel, Applet, or Dialog).
JFrame	Swing JFrame (top-level application window).
JDialog	Swing JDialog (modal or non-modal window for user feedback).
JApplet	Swing JApplet (embeddable program run by a Web browser or other applet viewer).
JPanel	Swing JPanel (lightweight container for holding parts of an interface—within any container, such as a JFrame, JPanel, JApplet, or JDialog).
JInternalFrame	Swing JInternalFrame (lightweight object with many of the features of a native Frame, used as a "window within a window" within a JDesktop-Pane).

Note that the table lists only the basic types which differ both in design-time look and in the code generated for the form's class. Customized forms (for example, a dialog with **OK** and **Cancel** buttons) and customized user classes (derived from one of the container classes above) can also be created and then saved as new templates. See "Creating your own templates" on page 150.

Working with components

Once you have created a new form, the first thing you probably want to do is to add components to it in order to create functionality. This section describes how to add and modify components in the Form Editor.

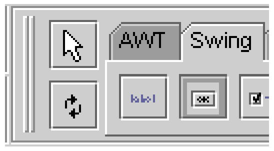


For more information on individual Swing components, see Sun's Swing tutorial at http://java.sun.com/docs/books/tutorial/uiswing/components/components.html.

Adding new components

The easiest way to add components is by using the Component Palette. The Component Palette is a toolbar on the Main Window, which holds commonly used visual components that you can add to forms. You can add a component merely by clicking on the component in the Palette and then clicking on the form in the Form Editor window. You can also add your own components to the Component Palette. See "Customizing the Component Palette" on page 155 for more information.

Add mode



To add a component using the Component Palette:

1 Choose a component in the Component Palette by clicking on its icon. (The add/selection mode icon shown in the figure above will become unselected, showing that the Form Editor is in “add mode”.)

2 Click on the Form Editor window to add the component to the form. You can add multiple components of the same type by holding down the SHIFT key and clicking multiple times.

The Palette automatically changes to selection mode (the add/selection icon becomes selected) after the component has been added to the form (and the SHIFT key has been released).

Depending on the layout used, you can add a component (or move it by dragging it) to a specific position (Border and Absolute layouts) and/or set the default size of the component by dragging the mouse on the selection handles (Absolute Layout).

To add a component using the Explorer:

1 Under `Repository` in the Explorer, find the class of the JavaBean you want to add.

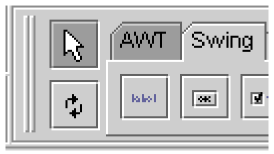
2 Right-click on the class and select **Copy** from the popup menu.

3 Choose **Paste** from the popup menu of either the Component Inspector or the Form Editor.

Selecting components

You can select components in the Form Editor either directly in the Form Editor window when in selection mode (the default mode, shown in the figure below) or by selecting component nodes in the Component Inspector's tree. To select multiple components when clicking on the Form Editor window, hold down the CTRL key as you select them.

Selection mode



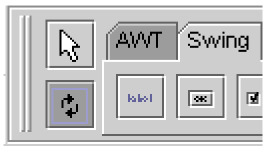
The Component Inspector displays the current selection on the active Form Editor window. Its property sheet displays the properties of the selected component—or, if more than one component is selected, their common properties are displayed. To select a component in the Component Inspector's tree, click on it. To select multiple components, hold down the CTRL key and click each one. You can also select a consecutive group of components by clicking on the first one, holding the

SHIFT key and then clicking on the last component in the group.

Connection mode

In connection mode, you can start the Connection Wizard, in which you can link two components together with an event.

Connection mode



See "Using the Connection Wizard" on page 104 for more information.

Copying components

To copy or move components:

1 Select the item in the Component Inspector or the Form Editor window (hold down the SHIFT or CTRL key to enable selecting of more than one item).

2 Right-click the item(s) and select **Cut** or **Copy** from the popup menu.

3 Choose the destination container in the Component Inspector or the Form Editor window and select **Paste** on its popup menu.

The components will be copied with all properties and events intact – although events are maintained only if copying within the same form.

Reordering components

A **container** is a Java visual component (e.g. a Panel) that can contain other components. (Examples of containers are the AWT Dialog, Frame, and Panel, as well as the Swing JPanel, JDialog and JFrame.) The order of components within their parent container may determine the order in which they appear visually. For some layouts (such as Flow Layout), modifying the order is the only way to modify the way components are laid out.

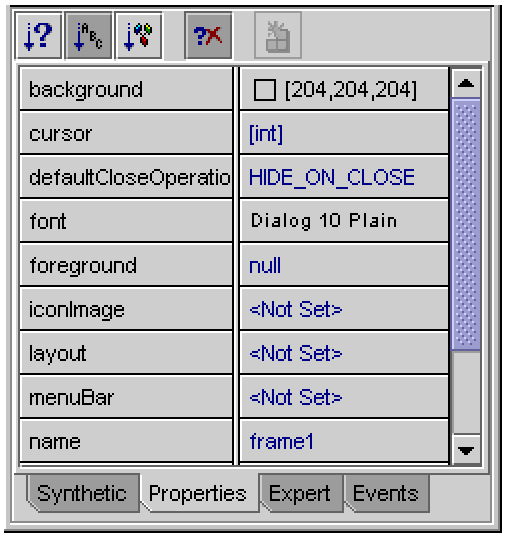
To change the order of components in the Component Inspector, either:

◊ Right-click on the component you want to move, and use the **Move Up** or **Move Down** items in its context menu; or

◊ Right-click on the parent container and choose **Change Order** from its context menu. You will get a popup menu for setting the order of all subcomponents in the container.

Property Sheet pane in the Component Inspector

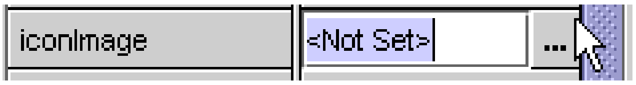
The lower pane of the Component Inspector displays the property sheet for the object selected in the upper pane. The property sheet contains up to four categories, represented by the tabs **Synthetic**, **Properties**, **Layout**, and **Events**.



background	☐ [204,204,204]
cursor	[int]
defaultCloseOperatio	HIDE_ON_CLOSE
font	Dialog 10 Plain
foreground	null
iconImage	<Not Set>
layout	<Not Set>
menuBar	<Not Set>
name	frame1

Depending on the property, properties can be edited by:

- clicking on the current value and typing in a new one (you must press the ENTER key for the new value to be accepted)
- double-clicking on the property name or value to toggle a boolean
- clicking on the value and then selecting from the drop-down list that activates.

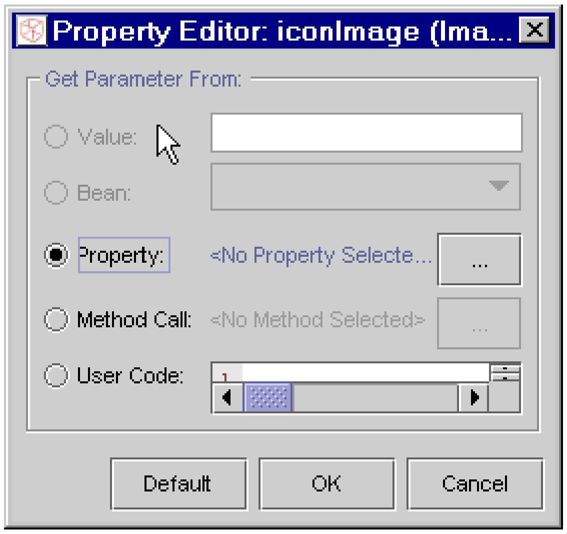


iconImage	<Not Set> ...

- clicking on the value and then selecting the **...** button that appears. Clicking on this button invokes a custom property editor dialog.

Custom property editors for component properties

When you invoke the custom property editor for a property of a form component, the Property Editor dialog that appears may have more than one tab. For example, properties on the **Properties** tab have an Form Connection property editor associated with them (see “Using the Form Connection property editor" on page 106).



Reverting to the default value for a property

If you have changed a value that you later decide you would like to change back to its default, right-click on the property name and select **Set Default Value** from the context menu. If the value was changed in a custom property editor, you must select the **Default** button in the custom property editor to revert to the default.

Working with layouts

A layout manager is a special Java class which manages the positions and sizes of components in a container (Frame, Panel, or any other visual component that can contain other components).

The Forte for Java Community Edition 1.0 Form Editor has advanced support for layout managers. In addition to absolute positioning, complex forms based on Flow Layout, Border Layout, Card Layout, Grid Layout, GridBag Layout or Swing's Box Layout are supported. (See "Standard layout managers" on page 94 for descriptions of these layout managers.)

Various containers come with different predefined layout managers:

- Frame, JFrame, Dialog, JDialog, and JInternalFrame have the Border Layout by default.
- Panel and JPanel have Flow Layout as the default layout.

Layout managers do not apply to all containers. For example, ScrollPane, JScrollPane, JTabbedPane, JLayeredPane/JDesktopPane, and JSplitPane have their own default layouts which cannot be changed.

Setting and changing layout managers

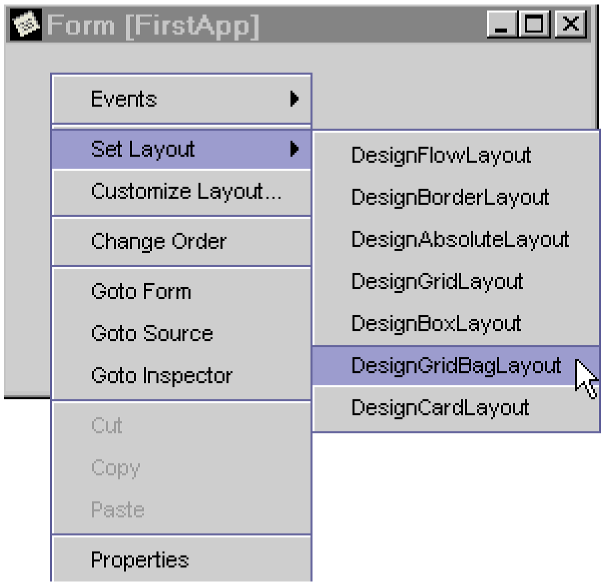
To change the layout manager:

1 Select the desired layout in the **Layouts** tab of the Component Palette.

2 On the form, click inside the container whose layout you want to change.

Or:

1 Right-click on the target container – either an empty part of the form in the Form Editor window or the node for the container or current layout in the Component Inspector.



2 Change the layout using the **Set Layout** submenu.

Note: When you change layouts, the IDE remembers the constraints used on the discarded layout. Therefore, if you change the layout back, it looks the same as it last looked in that layout. The only exception to this is when you switch from Absolute Layout to GridBag Layout. When you switch from Absolute to GridBag, the GridBag constraints are created so that the GridBag Layout looks as close as possible to the previous Absolute Layout. See “Standard layout managers” on page 94 for more information on the various layouts.

Setting layout properties and constraints

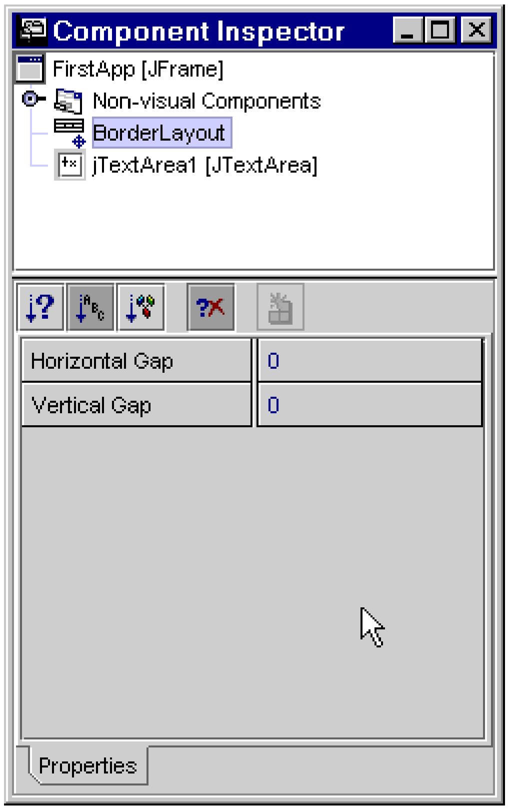
You can modify the appearance of a container on your form by setting general properties for the layout manager and constraints for the individual components in the layout.

Layout properties

The layout properties allow you to modify the layout manager's overall settings, such as the horizontal and vertical gap and alignment. The properties vary depending on the layout manager. Some layout managers do not have any properties.

To set layout properties:

◊ Select the layout manager's node in the Component Inspector. The layout properties will then appear in the Property Sheet pane.



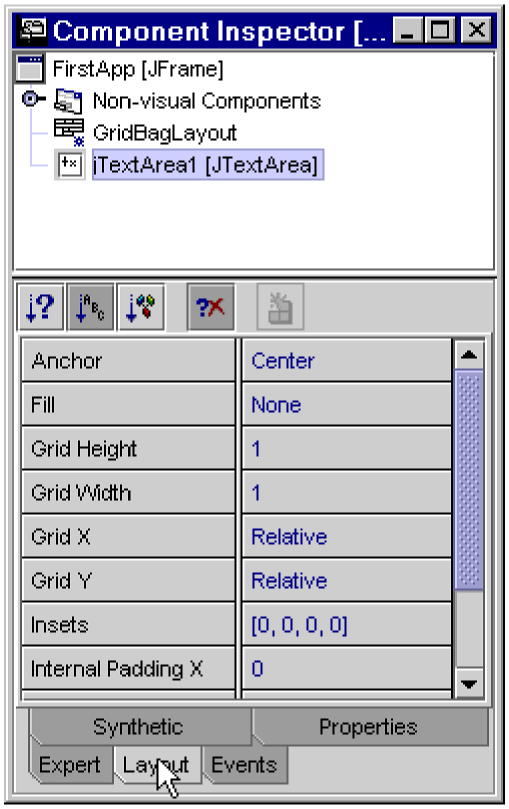
Horizontal Gap	0
Vertical Gap	0

Constraints properties

These are the layout properties specific to each component managed by the layout manager. For example, when Border layout is used, the `Direction` property is set according to which of the five parts of the form the component resides in (the possible values being `North`, `South`, `West`, `East` and `Center`).

To view or set constraints properties:

◊ Select the component in the Component Inspector and choose the **Layout** tab.



Anchor	Center
Fill	None
Grid Height	1
Grid Width	1
Grid X	Relative
Grid Y	Relative
Insets	[0, 0, 0, 0]
Internal Padding X	0

Standard layout managers

Border Layout

A Border Layout allows components in a container to be arranged in five positions: `Center` (which expands to fill any empty space left in the other four parts of the component) and along the four sides `North`, `West`, `South`, and `East`. Components in the `North` and `South` positions automatically expand horizontally to take up the whole space, and the `West` and `East` components automatically expand vertically. The horizontal and vertical gap properties control spacing between the components.

Flow Layout

A Flow Layout arranges components in a container like words on a page: it fills the top line from left to right, then does the same with the lines below. You can specify the alignment of the rows (`left`, `center`, `right`). You can also set the horizontal spacing between components and the vertical spacing between rows.

Grid Layout

The Grid Layout divides its Container into a configurable number of rows and columns. Like the Flow Layout, components are added to the grid from left to right and top to bottom. But unlike Flow Layout, the dimensions of the grid are fixed; individual components aren't allowed to have different sizes.

Card Layout

Think of a Card Layout as a deck of cards. Only one component ("the top card") is visible at a time and each component ("card") is the same size. In the Properties sheet, you can set the `Current Card` property to `card1`, `card2`, etc. In code, the methods `show`, `next`, `previous`, `first`, and `last` can be used to select cards. For example, the following code selects the next card in `panel1` when `button1` is clicked:

```
private void button1ActionPerformed(java.awt.event.ActionEvent
     evt) {
  // Add your handling code here:
  CardLayout cards = (CardLayout)panel1.getLayout();
  cards.next(panel1);
}
```

Also see the JTabbedPane component. It implements a similar layout with a labeled tab for each component.

GridBag Layout

GridBag Layout (not to be confused with Grid Layout) lets you set almost any layout you want using a complex set of constraints. You can create a GridBag layout one of the following three ways:

- By using Forte for Java's **Customize Layout...** command on a form to invoke the customizer dialog, which allows you to visually adjust the placement and constraints of the components. See "Using the GridBag customizer" on page 97 for more information.
- By setting the constraints yourself under the **Layout** tab of the Property Sheet of every component within the layout.
- By creating an Absolute Layout and then switching to GridBag to have the constraints code generated automatically. If you use this option, you may want to go back and polish up the details using the two previous options.

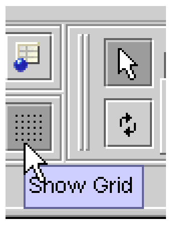
GridBag Layout is particularly useful for multi-platform Java applications, because it allows you to create a free-form layout that maintains a consistent appearance, even when the platform and look and feel change.

Box Layout

The Box Layout is part of the Swing API. It lets multiple components be arranged either vertically (along the Y axis) or horizontally (on the X axis). Components don't wrap – even when the container is resized. By default they are arranged from left to right or top to bottom in the order they are added to the container. See “Reordering components” on page 89 for changing the order of the elements. Box Layout is similar to Grid Layout, except that it is one-dimensional (i.e. there can be multiple components on one axis but not both).

Absolute Layout

Absolute Layout is a design aid that allows you to place components exactly where you want to in the form and move them around by dragging. Ease of use makes this layout particularly useful for making prototypes. You do not have enter any property settings and there are no formal limitations within the layout. It is a substitute for “null” layout and provides and places components in absolute positions more cleanly.



You can have a grid displayed on the Form Editor window when using Absolute Layout by pressing the **Show Grid** icon.

Important: Absolute Layout is not recommended for production applications. The fixed location of components in the form does not change, even when the environment changes. Therefore significant distortions in appearance can occur when such an application is run on a different platform or using look and feel different than the one in which it was created. If you design a form using Absolute Layout, it is recommended that you switch the layout manager to GridBag Layout and then fine tune it before you distribute the application.

Note: Absolute Layout is not in the standard Java layout sets. Rather, it is provided with the IDE and can be found at `com.netbeans.developer.awt.AbsoluteLayout` and `com.netbeans.developer.awt.AbsoluteConstraints`. If you do keep Absolute Layout in your application, these two classes must be distributed with it. They are ready for deployment in the `AbsoluteLayout.zip` file (in the `%FORTE4J_HOME%/lib/ext` directory). You can freely redistribute this archive with applications developed with the Forte for Java IDE. The source code is also redistributable and is available in `%FORTE4J__HOME%/sources/com/netbeans/developer/awt`.

Null Layout

You can also disable Absolute Layout and design forms with a null layout if you wish.

To use null layout:

1 Under the **Project Settings** tab of the Explorer, select the `Form Objects` node and switch the value of its `Generate Null Layout` property to `True`.

2 When designing your form, select the Absolute Layout layout manager.

Using the GridBag customizer

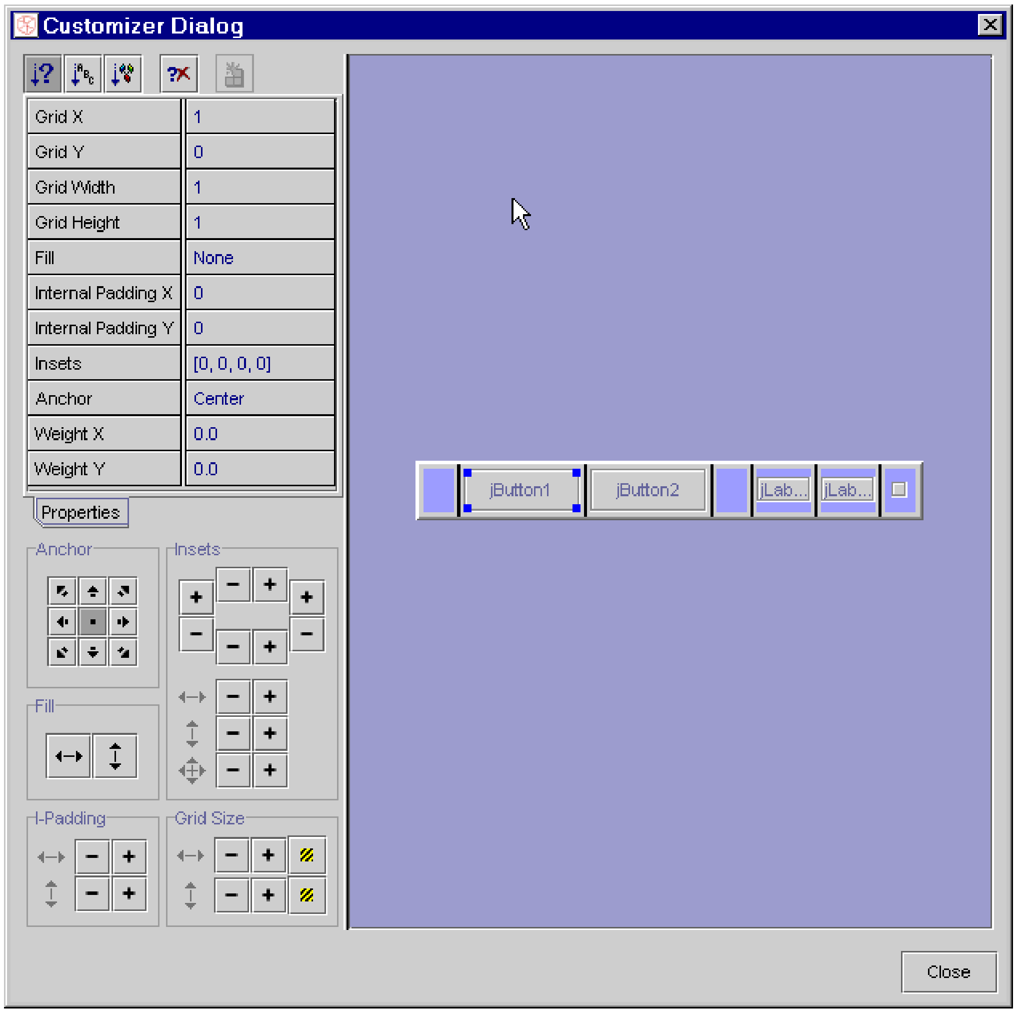
Forte for Java Community Edition 1.0 provides a customizer to simplify the creation of GridBag layouts.

To use the visual GridBag customizer:

1 Add the components you want to use and make sure that you have GridBag Layout set.

> **Note:** It is helpful to sketch out the way you want your layout to look before you open the customizer.

2 Right-click on the `GridBagLayout` node in the Component Inspector and select **Customize Layout...** from the context menu, or select the **Customize** button on the Property Sheet pane. The Customizer Dialog will open with a property sheet for GridBag constraints in its left pane and a rough depiction of the placement of your components on the form in its right pane.



Grid X	1
Grid Y	0
Grid Width	1
Grid Height	1
Fill	None
Internal Padding X	0
Internal Padding Y	0
Insets	[0, 0, 0, 0]
Anchor	Center
Weight X	0.0
Weight Y	0.0

3 You can reposition the components in the right pane by dragging them. As you drag, the component, its `Grid X` and `Grid Y` properties will change to reflect its new position. (Be aware, however, that this panel only serves as a rough guide and does not reflect the absolute positions

of the components. The position of each component is largely governed by other constraints set in the left pane. The Form Editor window more closely reflects how the components will look.) You can also change these values manually in the left pane (remember to press ENTER after entering the value).

4 Once you have the approximate position of the components, you can adjust the other constraints of each component in the left pane. Follow these steps (in whatever order you prefer) to position each component:

a. Select a component in the right pane.

b. Further adjust its horizontal and vertical position if necessary by setting its X and Y grid positions.

   Note: When you directly enter values for the constraints, remember to press ENTER to make the change (otherwise the property reverts to its previous value).

c. Adjust the `Grid Width` and `Grid Height` parameters to determine how many grid positions are allocated for the component in each direction. The values of `Remainder` (allocates the rest of the space in the given row or column for the component) and `Relative` are also available.

   You can also adjust these settings with the Grid Width buttons. Pushing the right-most button with yellow shading in each group sets the value to `Remainder`.

d. Adjust the weight settings of the component to determine how much space it should be given relative to that of the other components. For example, a component with a `Weight X` value of .5 has twice as much horizontal space allotted to it as a component with a value of .25 for this parameter. The sum of the values of the components in a given row or column should not add up to more than one. When a form is resized, these settings affect which components are resized (and by how much). Components with a value of 0 for one of these parameters retain their preferred size for that dimension.

e. Adjust the insets for the component. The insets determine the amount of free space on each of the four sides of the component.

   You can enter numbers for these manually or use the inset buttons on the bottom part of the left pane. These buttons are divided into four sets. The top group allows you to increase and lower the inset for each side separately. The second group allows you to change the left and right insets simultaneously. The third group lets you change the top and bottom insets simultaneously. The fourth group allows you to change all insets simultaneously.

   As you change the insets, you will see the inset area marked by yellow change in the right pane.

f. The internal padding settings allow you to increase the horizontal and vertical dimensions of the component. You can adjust these by directly entering numbers for the properties or by using the internal padding buttons.

   As you adjust these constraints, you can see the selected component in the right pane expand or contract vertically and/or horizontally, according the changes you make.

g. The `Fill` constraint allows you to choose whether the component will use all of the vertical and/or horizontal space allocated to it. Any space allocated to a component that the component does not fill is marked with blue in the right pane.

h. The `Anchor` constraint allows you to place the component in one of nine positions within the space allocated to it (`Center`, `North`, `NorthWest`, etc.). This setting has no effect if there is no free space remaining for the component.

Support for custom layout managers

It is also possible to use custom layout managers in the Form Editor. Any layout manager with a default constructor and which does not use constraints for adding components can be used in the Form Editor when designing.

Important: When adding custom layout managers that do not follow these criteria, you must follow the specifications of the Forte for Java Layout Managers API in order to be able to use the layout manager in the Form Editor. See the NetBeans website for more information on obtaining and using the Forte for Java Layout Managers API.

To install a custom layout manager

1 Add the directory or JAR archive containing the layout manager’s source or class to the Repository.

2 Install the custom layout manager in the Component Palette by copying the layout manager’s class from the Repository to a Palette category under `Global Settings / Component Palette`.

Working with source code

The Editor window displays the code for the active opened form. The source code is always synchronized with the visual appearance of the form. Every change made in the Form Editor window or the Component Inspector is immediately reflected in the source code.

Non-editable blocks

Some parts of the source code generated by the Form Editor are not editable manually. The background of non-editable text is shaded. Non-editable text includes:

- the block of variable declarations for the components on the form;
- the method `initComponents()`, in which all the form initialization is performed (this method is called from the form's constructor)
- header (and trailing closing brace) of all event handlers

Though guarded text generated by the Form Editor is not editable manually, you can affect the way it is generated using the Form Connection custom property editors for the components in your form.

See "Using the Form Connection property editor" on page 106.

Tip: If you want to perform some additional initializations of a form's components, you can do it in the constructor after the call to the `initComponents ()` method. For example, here you can initialize a button group or set component labels from resource bundles.

External modifications

Forms are stored in two files:

- a `.java` file, which contains the (partly) generated Java source
- a `.form` file, which stores layout manager properties, the properties and layout constraints of JavaBean components on the form, and other information. This file does not need to be distributed with your application. It is merely used to display the form in the Form Editor.

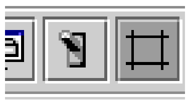
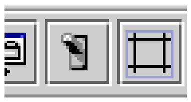
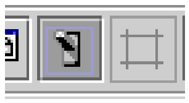
You can edit the `.java` files using external editors (not simultaneously while the form is being edited in the IDE), with the following exceptions:

- Do not modify the content of the `initComponents ()` method. The body of this method is always regenerated when the form is opened.
- Do not remove or modify any of the special comments placed in the source by the IDE's Form Editor (`// GEN-...`). They are required for the form to open correctly (These comments are not visible inside Forte for Java's source editor.)
- Do not modify the block of variable declarations for components on the form. They are always regenerated when the form is opened.

Form Editor modes

The Form Editor can be in one of three modes:

Table 5: Form Editor modes

Toolbar	***Description***
	*Design mode*. This is the default Form Editor mode. By clicking on the form you can select, add, or drag components. Note that in this mode, depending on the layout, the layout won't necessarily look the same as it does during run-time. This is because design would be difficult if every layout manager worked exactly the same way at design time as it does during run time. For example, with some layout managers, when you drop two components next to each other, they could resize automatically and make it impossible to add a new component between the two. So, if you need to see the effect of a certain layout manager at run time, change the Form Editor from design mode to either real mode or test mode.
	*Real mode*. The Form Editor window displays the actual Java layout managers, so the forms looks the way they will during run-time. You can select or add components, but dragging components is disabled.
	*Test mode*. Similar to real mode, the form behaves exactly the same as during run-time. When clicking on the form, the mouse events are delivered to the actual components. Thus, for example, a button looks "pressed" when you click on it. This mode is suitable for checking the look and feel of the form without the need to compile and run it.

To switch between design and real mode, use the **Design Mode** icon (the square with a pound sign inside) from the toolbar. If the icon is selected (as shown in the design mode example above), you're in design mode. Otherwise, you're in real mode.

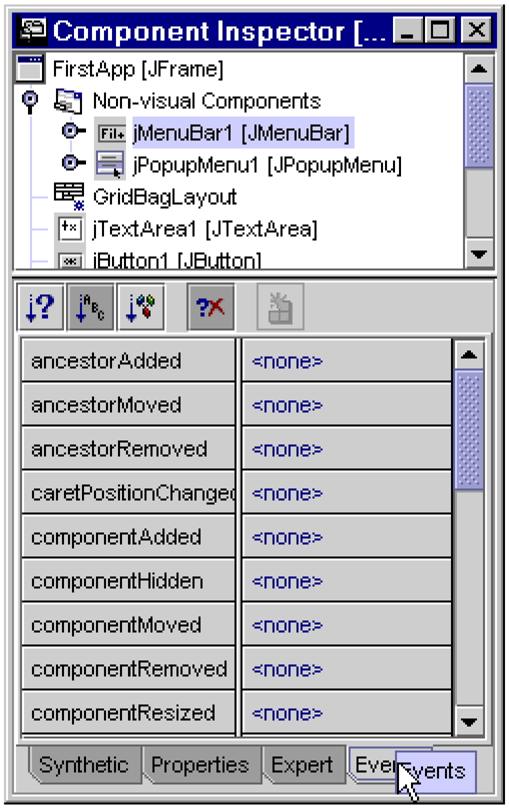
Switching to test mode is done using the **Test Mode** icon on the toolbar; this is a toggle button, which means you click to enable it and click again to disable. Note that switching test mode on disables the **Design Mode** icon; to enable it again, test mode must be switched off.

These modes are kept separately for each form.

Note: When test mode is switched on, the form is resized to its preferred size – this is how it looks if it is not explicitly resized by calling `setSize()` or `setBounds()`.

Events

Events are managed through the component's **Events** tab in the Component Inspector, which gives a list that matches each event fired by an object with its corresponding handler method. There is also an **Events** entry on object popup menus in both the Component Inspector and Explorer.



FirstApp [JFrame]
Non-visual Components
jMenuBar1 [JMenuBar]
jPopupMenu1 [JPopupMenu]
GridBagLayout
jTextArea1 [JTextArea]
jButton1 [JButton]

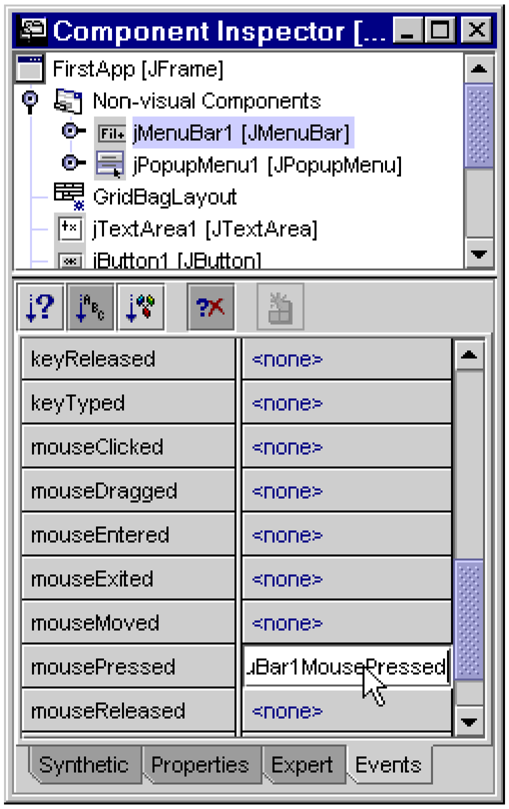
ancestorAdded	<none>
ancestorMoved	<none>
ancestorRemoved	<none>
caretPositionChange	<none>
componentAdded	<none>
componentHidden	<none>
componentMoved	<none>
componentRemoved	<none>
componentResized	<none>

A new component does not have event handlers defined; the values are all `<none>`.

To define an event handler using the property sheet:

1 Select the **Events** tab in the form object's property sheet (either in the Component Inspector or Explorer).

2 Click on the value (it should be `<none>`) of the one of the events in the list.



keyReleased	<none>
keyTyped	<none>
mouseClicked	<none>
mouseDragged	<none>
mouseEntered	<none>
mouseExited	<none>
mouseMoved	<none>
mousePressed	Bar1MousePressed
mouseReleased	<none>

3 Select a name for the event by:

- Pressing ENTER on your keyboard to have the default name automatically generated; or
- Typing a new name over the default name and pressing ENTER.

After you press ENTER, the code for the listener and the (empty) body of the handler method will be generated. If you don't press ENTER, your changes will be ignored.

To define an event handler using the popup menu:

1 Right-click on the form object in the Explorer.

2 Select **Events** and then move through the two submenus to select the event. The default name will be given to the event.

If multiple events are of the same type (for example, `FocusGained` and `FocusLost` are both of type `java.awt.event.FocusEvent`), then you may use the same handler for all of them. For example, you could set both `FocusGained` and `FocusLost` to use the `button1FocusChange` handler. You can also have the same event on several components share a handler.

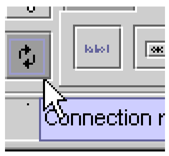
When you enter a new name for an existing handler, the code block is automatically edited to use the new name. When you delete a name, the code block is deleted. If more than one handler uses the same name (and the same block of code), deleting a single reference to the code will not delete the code; only deleting all names will delete the corresponding code block, and a confirmation dialog will be displayed first.

Note: If you delete an event but don't delete the event handler when prompted by the confirmation dialog, an orphaned handler will remain in the guarded text. If you wish to delete the orphaned handler, you will have to add another event, attach it to the handler, delete that event, and then agree to delete the handler in the confirmation dialog.

Using the Connection Wizard

The Connection Wizard allows you to set events between two components without having to write much (or any) code by hand. When you select two components to be "connected", the first is the one that will fire the event, and the second one will have its state affected by the event.

To start the Connection Wizard:



1Switch to connection mode on the Component Palette (see "Connection mode" on page 89).

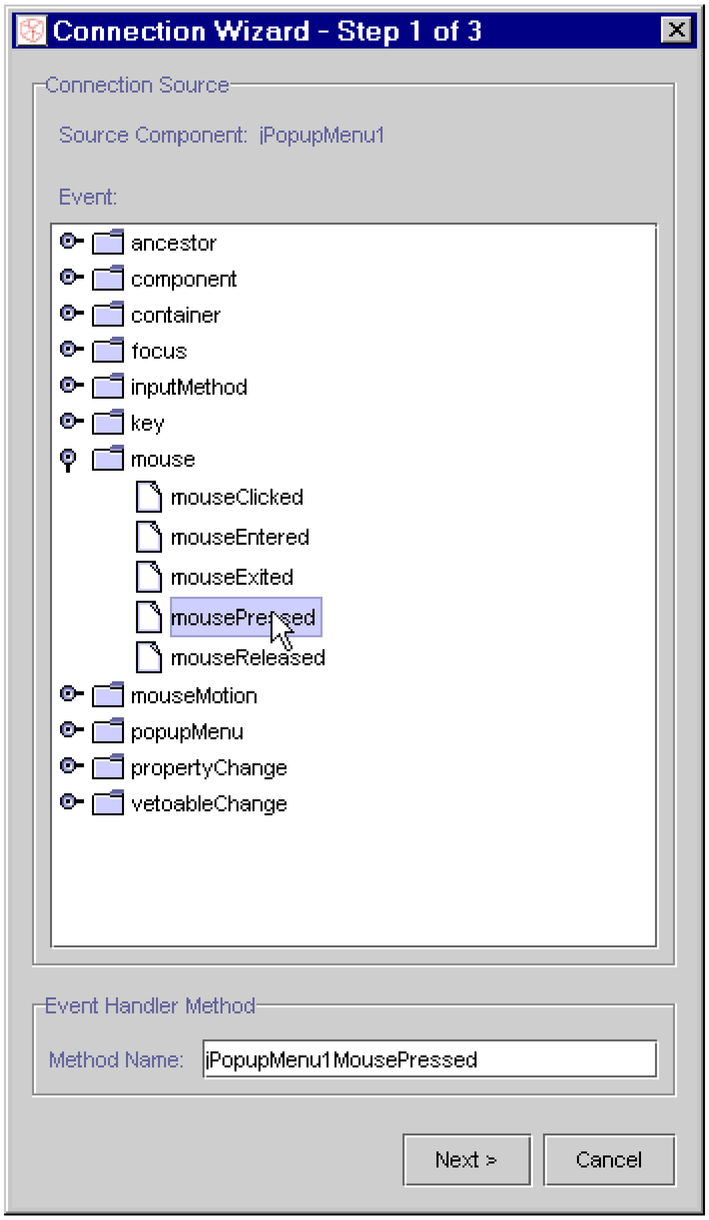
2Next, click on the two components (first the component with the source event and then the target component). You can click on the form—or click in the Component Inspector (which allows you to also use non-visual components as parts of the connection).

The Connection Wizard opens to connect the selected components.

The Connection Wizard allows you to set the connection parameters in two or three steps:

Connection Source

The first step allows you to select the event of the source component on which the operation is to be performed and the name of event handler method generated.



Note: If you select an event which already has an event handler attached (the name of the event handler is visible in the tree of events after the event name – like `actionPerformed [button1ActionPerformed]`), when clicking **Next**, you will be asked if you want to delete the old event handler. If you answer yes, the old event handler is replaced with the connection code when the Connection Wizard is finished.

Connection target

The second step allows you to specify the operation which is to be performed on the target component. There are four options which can be changed using the radio buttons along the top of the dialog:

- **Set Property** – allows you to set a property on the target component. In the next step you specify the property value.
- **Bean** –
- **Method Call** – allows you to call a method on the target component. In the next step you specify the parameters for the method call.
- **User Code** – allows you to write the code by hand. When you select this option, the **Next>** button changes to **Finish**, meaning there are no settings remaining in the wizard and the target component is ignored by the wizard. The cursor in the editor is then placed into the new event handler allowing you to write the handling code by hand.

Selecting property value or method parameters

If you have selected **Set Property** or **Method Call** in the previous step, the third step will allow you to specify the values for the target property or parameters for calling the target method. If you selected the **Method Call**, the dialog will display a list of all parameters types as tabs, where each tab represents a single method parameter.

On each tab, you can set the source from which the value (of the specified type) is acquired:

- **Value** – This option is available only if the value is a primitive type (`int`, `float`, etc.) or a `String`. You can enter the value into the text field.
- **Property** – This option allows you to acquire the value from a property of a component on the form. If you click on the **...** button, a dialog will appear which will let you select the component and its property. Note that only properties of the correct type are listed.
- **Method** – This option allows you to acquire the value from a method call of a component on the form. If you click on the **...** button, a dialog will appear which will let you select the component and its method. Note that only methods which return the correct type and which do not take any parameters are listed.
- **User Code** – This option allows you to write user code which is used as the parameter for the setter of the selected property or a parameter of the selected method call.

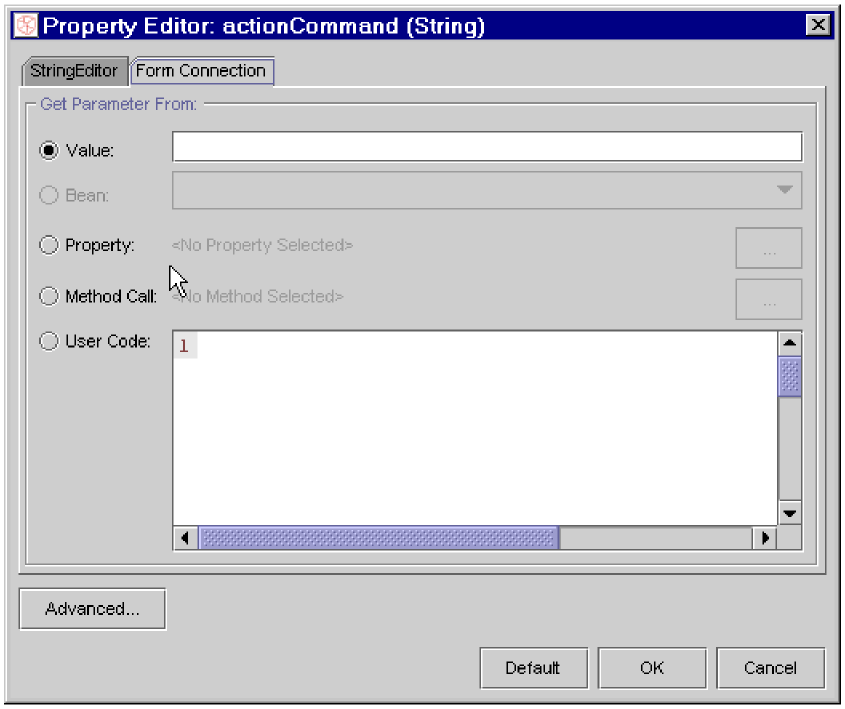
If you click **Finish**, a new event handler will be generated with code reflecting the settings entered in the second and third step.

Using the Form Connection property editor

Forte for Java Community Edition 1.0 also provides a connection feature that allow you greater control over the initialization code that is generated for properties of form components. You can associate a method, a property, or your own code with the property. This property editor is particularly useful, since initialization code is guarded text in the Editor and cannot be manually edited.

To use the Form Connection property editor:

1. With the Component Inspector open on a form, select the property (under the **Properties** tab that you would like to modify the initialization code for.
2. Press the **...** button to invoke the custom property editor multi-tab dialog and select the **Form Connection** tab.
3. In the dialog, select the type of code you would like to add (**Value**, **Bean**, **Property**, **Method Call**, or **User Code**).



If you select **Property** or **Method Call**, press the **...** button to invoke a dialog with a list of properties or methods available.

4. Once you have made your selection, click **OK**.

Pre- and post-initialization code

In the Form Connection property editor, you can also write code to be placed before or after the initialization code for the modified property.

To place code before or after a property's initializer:

1. In the property's Form Connection property editor, click **Advanced...** to bring up the Advanced

Initialization Code dialog.

2 Click either or both of the radio buttons (for **Generate pre-initialization code** or **Generate post-initialization code** and enter the code you want to have generated in the appropriate text fields.

Synthetic properties

Each component also has synthetic properties which you can set under the **Synthetic** tab in the Property Sheet pane in the Component Inspector. These are:

- **Code Generation** – allows you to choose between generating standard or serialization code for the component.
- **Serialize To** – allows you to set the name of the file for the component to be serialized to (if it is serialized).
- **Use Default Modifiers** – If `True`, the component's variable modifiers will be generated according to the default modifiers set in the `Variables Modifier` property set in the property sheet for `Project Settings / Form Objects` in the Explorer. If `False`, the `Variable Modifiers` property will appear on the property sheet allowing you to set the modifiers.
- **Variable Name** – allows you to change the variable name of the component in the code.

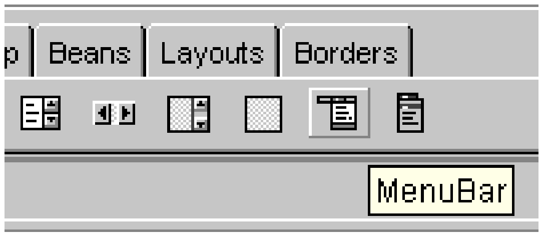
Menu editor

The Form Editor can be used to create and modify menus. The AWT and Swing variations of both `MenuBar` and `PopupMenu` are supported.

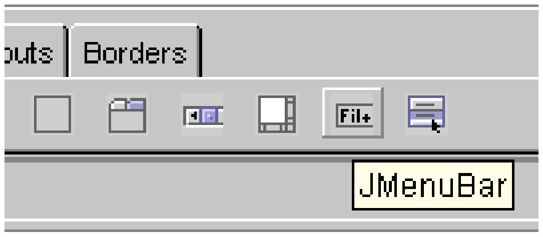
Creating a menu bar

To create a new menu bar, click on an AWT or Swing menu bar component in the Component Palette, as shown below. Then click anywhere on the form on which you want to add the menu. The menu will appear under the `Non-visual Components` list in the Component Inspector (since it cannot be manipulated by the layout manager).

Creating a menu bar



If this is the first menu bar you have added to this form, it will also appear visually on the form. Please note that only the AWT `MenuBar` can be used as the main menu for AWT forms. The same is true for Swing forms and the `JMenuBar`. It is possible to add the AWT `MenuBar` to Swing forms and vice-versa, but you will not be able to use it as the displayed main menu for the form.



You can add multiple menu bars to one form, but only one of these can be used as the current menu bar. (You can write code to switch the current menu bar when the form is running.) To change the current menu, select the form in the Component Inspector and edit its `Menu Bar` property. You can also switch the menu bars in user code while the form is running.

Adding menus to the menu bar

Newly created menu bars come with one menu.

To add more menus:

1 Right-click on the menu bar under the `Non-Visual Components` node of the form in the Component Inspector.

2 Select **New Menu** (or **New JMenu** if you are using Swing components) from the popup menu.

Creating a popup menu

To add a popup menu to a form:

1 Click on the popup menu icon under the **Swing** or **AWT** tab in the Component Palette.

2 Click anywhere on the form to place it.

The new menu will appear under the `Non-visual Components` list in the Component Inspector (since it is not laid out by the layout manager). You can use the same editing features as with a menu bar. To use the popup menu visually, write code like this:

```
    popupMenu1.show (component, 100, 100);
```

where you want to show the popup menu. The API documentation that comes with the Java 2 SDK has more details about displaying popup menus in its specifications for PopupMenu and JPopupMenu.

Adding menu items

The Component Inspector displays the hierarchy of menu bars and popup menus under `Non-visual Components`. Every menu (whether it is on a menu bar or it is a popup menu or submenu) starts with one menu item, displayed as a subnode.

To add new menu items, submenus, or separators to a menu bar or popup menu:

1 In the Component Inspector, select the menu you would like to add an item to.

2 Right-click on the menu and select **New** and then the item you want from the submenu.

Menu item events

You can use the same mechanism for assigning event handlers to menu items as with any other component. For menu items, you can also add the `ActionListener` event handler by actually selecting the item from the menu in the Form Editor window.

To add an event handler to a menu item:

1 Select the component in the Component Inspector and either use the popup menu or event properties to attach the event handler.

2 Use the Connection Wizard by clicking on the **Connect** icon and then (in the Component Inspector) clicking on the connection source and target. See “Using the Connection Wizard” on page 104.

Components with special support

Forte for Java Community Edition 1.0's Java-based architecture allows it to support interactive design with components including the AWT ScrollPane and the Swing JScrollPane, JTabbedPane, JTable, JList, JDesktopPane, and JSplitPane.

In general, these components are used the same way as other components. Forte for Java provides special support in the form of features such as custom property editors which simplify some of the more complex aspects of designing forms.

JScrollPane, ScrollPane

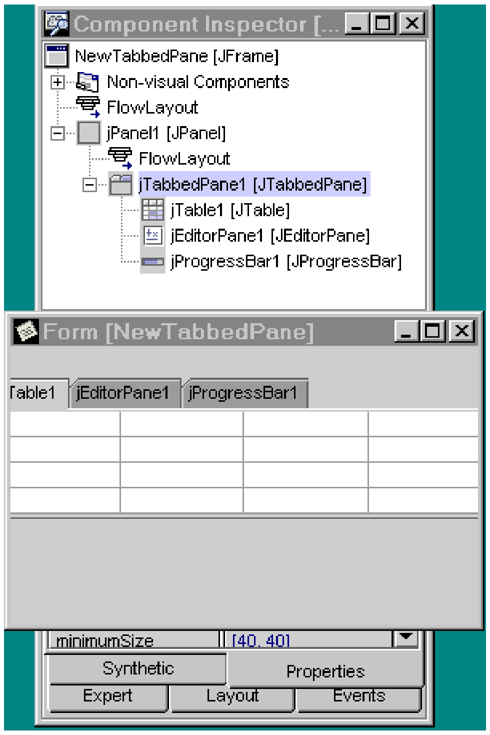
In many cases, AWT components pop up their own scroll bars if their content needs to scroll. An AWT ScrollPane is available, though, to add scrolling where it is needed. Swing components, however, must be added to a JScrollPane if the content is to scroll. This document will not cover other differences between ScrollPane and JScrollPane. See the Java reference materials and tutorials for these particular classes. We'll discuss JScrollPane as it is more frequently needed (and it is generally preferable to use Swing components).

To use a JScrollPane, choose it from the Component Palette and click on the Form Editor window. Then select the component to be added from the Palette to the JScrollPane and then click on the JScrollPane.

Remember that you can still select the underlying JScrollPane (for instance, to set its scrolling properties) by clicking on its entry in the Component Inspector.

JTabbedPane

A JTabbedPane manages multiple components (usually JPanels with their own layouts and sub-components) within its pane. The IDE automatically gives each component its own labeled tab. When the user clicks on the tab, its component is selected and comes to the front of other components.



The above figure shows a Form Editor window over the Component Inspector window. A JPanel has been added to the JFrame, a JTabbedPane has been added to the JPanel, and three components have been added to the JTabbedPane.

By default, each tab is named for its own component. However, we have renamed the tables tab and also added a tool tip that appears when the user holds the mouse over the tab. Notice that each component in the JTabbedPane has its own layout settings; this is where the tab parameters (`Tab Icon`, `Tab Name`, `Tab ToolTip`) are set.

Note: Be careful where you click when adding new components to a JTabbedPane. If you click on an existing tabbed component, your new component might be added to the existing component rather than to its own new tab.

JTable, JList

A JTable is a grid of cells, each cell having its own content. A JList is similar but with just one dimension. The special support for JTable and JList consists of a custom property editor that controls the number of rows (and columns), the titles, object types, etc.

To open the custom property editor for a JTable or JList:

1 Access the object's properties (e.g. through the Component Inspector).

2 Choose the **Properties** tab.

3 Click on the **...** button for the entry for the `model` property.

The JTable custom property editor has tabs for **Table Settings** and **Default Values**.

The JList custom property editor allows you to add, delete, and rearrange list items.

Note: These custom editors only provide the ability to edit simple models of relatively fixed structure. More demanding applications cannot use these visual editors. For live data with changing structure, create your own model in code and assign it to the table or list in the form’s constructor.

MDI Applications: Using JDesktopPane and JInternalFrames

Swing makes MDI (Multiple Document Interface – windows within windows) easy to implement. The model is like traditional computer windowing systems. In a windowing system, you have a desktop with windows on it. With the Swing MDI, you use a JDesktopPane with JInternalFrames on it. The user can position the JInternalFrames like windows on a traditional desktop as well as resize, close, and minimize them. Forte for Java Community Edition 1.0 lets you lay out and define all of this interactively.

You typically start with a JFrame and then add a JDesktopPane to it. To add JInternalFrames, select them from the **Swing2** tab of the Component Palette and click on the JDesktopPane. This adds internal frames with fixed structure. Alternatively, you can create separate forms for the types of frames you want, and then (construct and) add these in code to the desktop pane, thus giving you more flexibility.

You can add other components to the JDesktopPane such as a JTable or JSlider. However, these have their standard properties, and users can't manipulate them in the same way they can manipulate a JInternalFrame containing those components.

JSplitPane

A JSplitPane has two sides, each of which may have a component placed in it. A user can move the divider between the sides to make one side bigger and the other smaller. As with other special components of this type, Forte for Java allows you to manipulate it during design – for instance, to drag the divider. You can change the orientation of the divider from vertical to horizontal with the `Orientation` property (under the **Properties** tab of the property sheet). You may find that it's easiest to select the JSplitPane itself by clicking on its entry in the Component Inspector.

Adding JavaBeans to the IDE

JavaBeans™ Components is Sun's component model for Java and has been widely adopted by Java developers. Forte for Java automates the use and creation of individual JavaBeans (beans). It also enables the customization of beans, serialization of the customized state, and the use of the bean with specific settings. Both visual and non-visual JavaBeans can be easily added to the Component Palette and later used in visual design.

JavaBeans are distributed along with their manifests in JAR files. The IDE includes several installed JavaBeans, and offers the possibility of adding many more. Installing new JavaBeans is an easy way to expand the functionality of the IDE.

Standard method for adding JavaBeans

To add a JavaBean to the IDE:

1 Choose **Tools | Install New JavaBean** from the main menu.

2 In the dialog box, select the path and JAR file of your new JavaBean and click **Open**.

3 A list will appear with available JavaBeans. Select the ones you want and click **Install**.

> **Note:**If the beans you expect do not appear, the required attribute, `Java-Bean: True` is missing from the beans' entries in the JAR manifest. Please see the JavaBeans specification for details on proper packaging of JavaBeans.

4 In the Palette Category dialog that appears next, choose where in the Component Palette (under which tab) you would like to place the bean(s) and click **OK**.

You will then see the new bean(s) appear under the tab you indicated, ready to use in an application.

Some JavaBeans have their own icon. In cases where this is not true, the IDE assigns a default icon to the JavaBean. To see the name of each installed JavaBean, position the cursor over the bean and a tool tip will appear.

Alternate method for adding JavaBeans

You can also add beans from a JAR archive or from a local directory.

To add a bean from a JAR archive:

1 Either

- select **Tools | Add JAR** from the main menu; or
- right-click on `Repository` in the Explorer and choose **Add JAR** from the context menu; or
- right-click on `Project Settings / Repository Settings` and select **New | Jar Archive**.

2 Select the path and directory, click on the `.jar` file, and click **Mount**.

The archive then appears in the Explorer, under the Repository, and in the Object Browser.

3 If you are using the Explorer, expand the node of the archive or directory that you have just added and locate the JavaBean you want to add.

4 Right-click on the JavaBean in the Explorer or Object Browser and choose **Tools | Add to Component Palette** from the popup menu.

5 In the Palette Category dialog, select the tab under which you would like the bean to appear in the Component Palette and click **OK**.

The new JavaBean will appear in the Explorer and on the Component Palette in the Main Window under the category you assigned.

To add a bean not packaged in a JAR or ZIP file:

◊ Follow the previous procedure, but use **Add Directory** instead of **Add JAR** in step 1.

To add a bean to the IDE from the Repository:

◊ Follow steps 4 and 5 of the procedure for adding JavaBeans to the IDE.

Note: Using the Alternate Method you can also add JavaBeans that come as serialized prototypes (their filenames end with `.ser`).

Instead of using **Tools | Add to Component Palette**, you can copy and paste (using the **Edit** menu in the Main Window, context menus in the Explorer or Object Browser, or keyboard shortcuts) beans into the `Global Settings / Component Palette`. If you copy and paste using the main menu or a context menu, you can choose from up to four options from the **Paste** submenu. For more on these options, see “Package Popup Menu Commands” on page 138.

Automatic method

You can have a JAR file installed automatically each time the IDE is started by putting it into the `beans` directory under the root of the installation. By default, each bean is put into a Palette category with the same name as the JAR file.

You can control which beans are loaded and what Component Palette categories they are loaded into by adding a `beans.properties` file.

Chapter 6

Using the IDE

This chapter describes each part of the Forte for Java Community Edition 1.0 IDE in detail and tells you how you can customize it to your own specifications.

User interface

Just as Java is an object-oriented programming language, the Forte for Java Community Edition 1.0 user interface is *also* object oriented, providing multiple ways to accomplish most tasks.

The core of the user interface consists of the Main Window, Explorer, and Editor. The Form Editor window, HTML browser, Component Inspector, Object Browser, Debugger, Execution View, and Output Window are all parts that work closely with the core IDE. This default set of windows, workspaces and tools can be adjusted freely to match your preferences.

Look and Feel

The Forte for Java user interface offers different "looks" based on the platform being used. By default, the IDE launches in the Windows look and feel when running on a Windows platform and the Metal look and feel on all other platforms. It can then be adjusted to meet the developer's preferences.

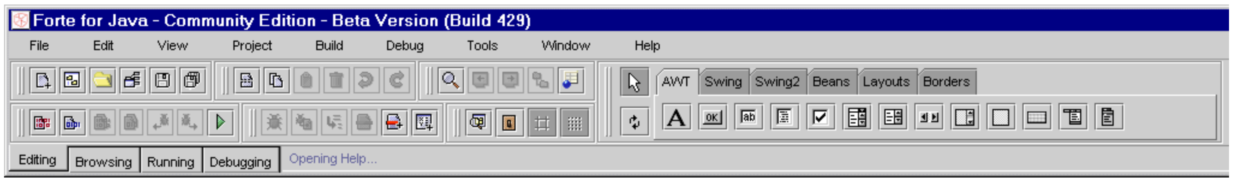
To change a look and feel:

◊ Go to **View | Look&Feel** on the main menu and select the look and feel you want from the submenu.

Main Window

The Main Window opens when Forte for Java is launched and remains open as long as Forte for Java is running. The Main Window can be viewed as the control center of the IDE. Most important operations and commands are accessible from this window. The Main Window can be broken into four separate groups of controls: the menus, the toolbars, the workspace tabs, and the status line.

Main Window with Metal look and feel



Menus and toolbars

A listing of all menu items and toolbar operations is given in "Main Window Menus" on page 172, and specific operations are mentioned in pertinent sections. Menu items and toolbar tools are context sensitive: they may be disabled (grayed out), depending on which window is currently active or which object is selected. Menu entries and toolbars can be re-ordered and customized. Keyboard shortcuts are shown in their corresponding menu items. See "Customizing menus and toolbars" on page 151 for more information.

Component Palette

The Component Palette (shown in the right half of the figure "Main Window with Metal look and feel" on page 117) is a special toolbar used in conjunction with the Form Editor to visually build forms. It consists of several tabs, each housing standard components and layouts.

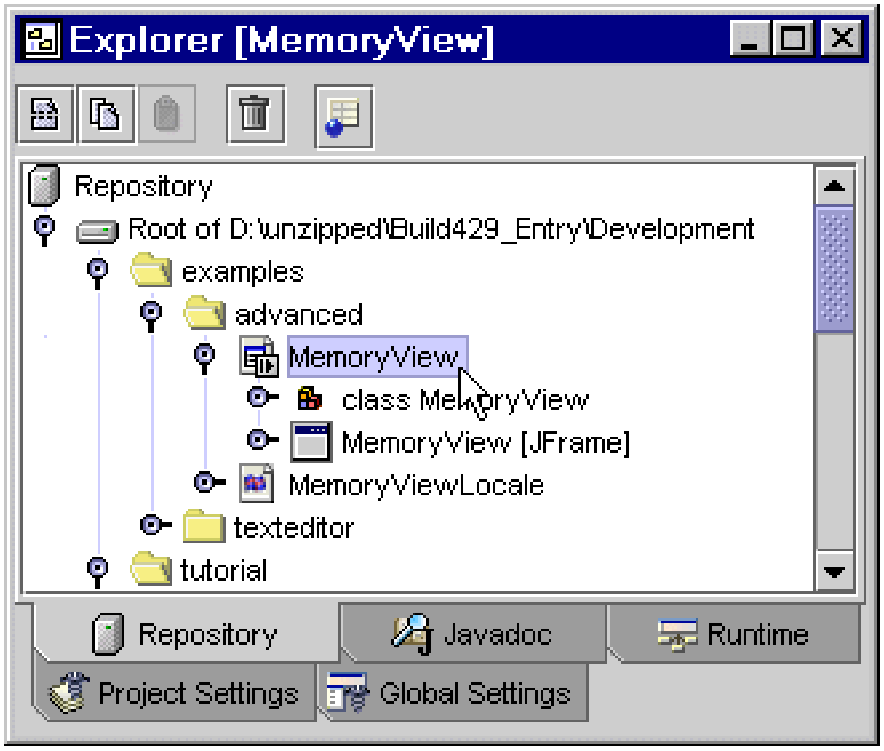
Workspaces

Forte for Java uses the concept of **workspaces** to manage windows and tools. On the lower left corner of the Main Window are four workspace tabs: **Editing**, **Browsing** (unless the Object Browser is not installed), **Running**, and **Debugging** (unless there is no Debugger installed). Each workspace contains a set of opened windows appropriate to specific stages of the development cycle. Clicking on each of these tabs "flips" between each workspace. By default, the IDE automatically switches to the Running Workspace when you execute an application and to the Debugging Workspace when you initiate a debugging session. For more information on managing workspaces, see "Workspaces" on page 130.

The Explorer

The Forte for Java Community Edition 1.0 Explorer gives a unified view of all objects and files and provides a starting point for many programming functions. Here, you can work with objects, organize your work, modify object properties, connect various data sources, and customize the Forte for Java environment.

The Explorer with Metal look and feel



Navigation

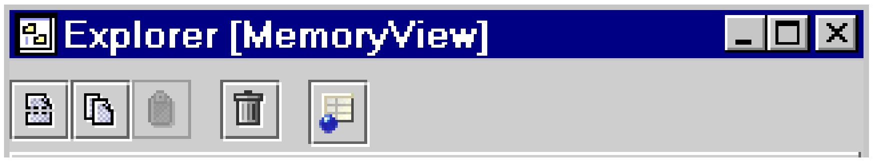
When you first start Forte for Java and open the Explorer, you will see a multi-tab window with tabs labeled `Repository`, `Runtime`, `Javadoc`, `Project Settings`, and `Global Settings`. Click on a tab to go to that part of the Explorer.

Navigating the hierarchy within each category is simple. Click on the expand button (⊶ for Metal, ⊞ for CDE Motif, and ⊞ for Windows) next to each item to expand the tree. Each tree node represents an object, and the object types are visually distinguished with icons. See "Object Types" on page 139 for an overview of available object types.

Right-click on any item to access its popup menu, which contains the context-sensitive set of operations (as well as access to the property sheet) available for that item.

Explorer toolbar

The Explorer toolbar presents icons for standard clipboard operations: **Cut**, **Copy** and **Paste**. It also includes a **Delete** icon and the **Toggle Property Sheet** toggle button. To see the name for each icon, hold your mouse cursor over it and a tool tip label will appear. As with the popup menu, the toolbar icons are context sensitive. For example, when a top-level node such as `Repository` is selected, the cut and delete functions are not available as the `Repository` cannot be deleted or moved.



Almost all objects in the Explorer hierarchy have some associated properties. The **Toggle Property Sheet** toggle icon in the Explorer toolbar can be selected to open the Property Sheet pane which displays the properties associated with each object. These property sheets can also be accessed via the **Properties** command on each object's popup menu.

Default operations

In the Explorer window, double-clicking on an item or selecting the item and pressing the ENTER key performs the default operation on that object: opening it, opening its property sheet, and/or expanding it.

The default operation varies for each object type. The most common operation is displaying the object, which means opening the selected object in an Editor window. For example, double-clicking on a Java object opens the source in the Editor. Double-clicking on a method or variable of a Java class, as displayed under a parent, opens the Editor window and positions the cursor on the line where that method or variable is declared. Double-clicking on a folder object, such as a top-level node, simply expands or collapses the sub-tree. Double-clicking on a Project Settings node opens a Property Sheet window for that item.

Property sheet

The property sheet displays (and, in the case of writable properties, allows you to edit) the properties of the items selected in the Explorer, whether they are files, components in the Form Editor, or the IDE's system objects. When multiple items are selected, the property sheet displays the properties that are common to all the selected items. If the items have multiple sets of properties, each set of properties is displayed on a separate tab.

Accessing the property sheet

There are several ways to access an object's property sheet.

To view the object's properties on the Property Sheet pane in the Explorer:

◊ Select the object in the Explorer hierarchy and then click the **Toggle Property Sheet** icon (as shown in the figure below).

To view an object's properties in the Property Sheet pane of the Component Inspector:

◊ Select the object in the Component Inspector.

To view the object's property sheet in its own window:

◊ Right-click on the object in the Explorer or Object Browser and select **Properties** from the popup menu; or

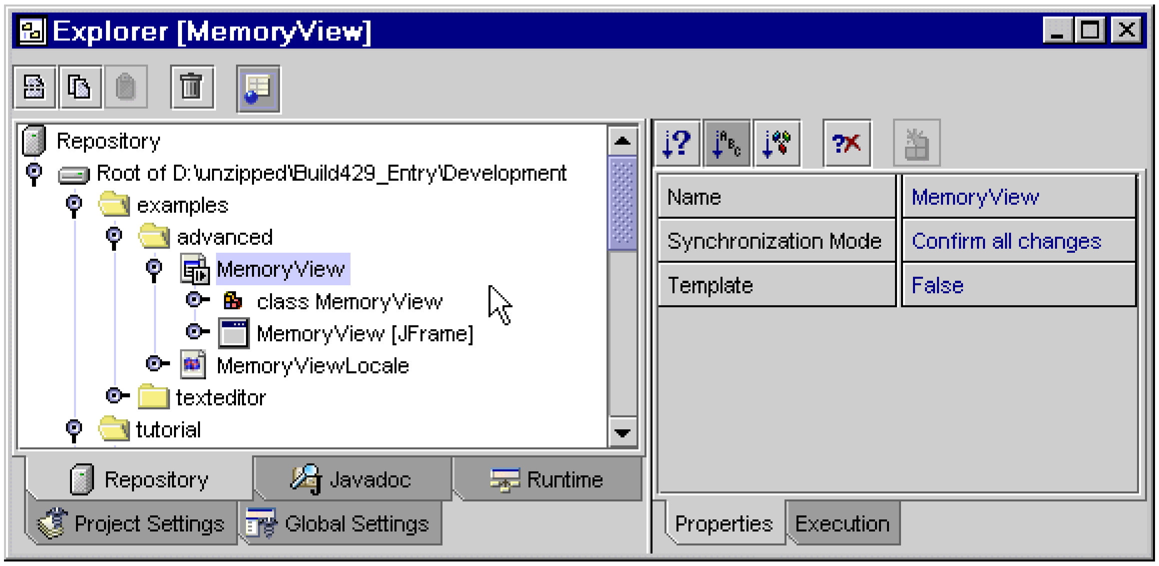
◊ Select the object and press ALT+1.

Contents of the property sheet

The Property Sheet pane consists of paired names and values. Each row of the property sheet represents a single property of one or more (if multiple objects are selected) items. A name/value pair may be disabled (grayed) if it represents a read-only property that is not connected with a custom property editor (a dialog tailored to the specific property allowing editing or input of more complex values).

Each property name has a tool tip label which gives a brief description of the property. To access the tool tip, rest the mouse over the property name for a moment until the tool tip appears.

Explorer with its Property Sheet pane



Name	MemoryView
Synchronization Mode	Confirm all changes
Template	False

For JavaBeans properties, the name of the property is derived from the display name of the bean property. The tool tip for each property name displays its accessibility information and a short description for that property. Accessibility details are shown in parentheses (`r/w`, `r/-`, `-/w` or `No property editor`) and depend on the property and its property editor (see Custom property editors below).

The value field either shows a property value as text or paints a representation of it (like the color preview for a color property). The tool tip for the value of the property displays the type of property, such as `java.awt.Color`. Clicking on this area switches the value field into input mode, allowing the value of the property to be changed. Double-clicking a name toggles the value if there are just two possibilities (for instance, changing `True` to `False`)—or, if multiple values are possible, the value advances to the next possible (for instance, `black` might change to `blue`, then to `red` and so on for each double-click). There are several ways to change the value, depending on the type of property. You may edit the value as text, select from a pull-down list, open a custom property editor (from the **...** button), or customize a painted preview directly in the painted area.

Property sheet toolbar

The first three icons on the property sheet toolbar are toggles for sorting (to leave unsorted, sort by name, and sort by type, respectively). The next toolbar button enables filtering of properties by accessibility (read/write) so that only writable properties are displayed. The right-most toolbar icon invokes the Customizer. The Customizer is a dialog which can be used for advanced customization of the whole object at once. This icon is context sensitive and is only available for certain objects.

Custom property editors

A **custom property editor** is a dialog, invoked by pressing the **...** button on a property value in a property sheet, specially designed for editing the value of that property. Custom property editors range from the simple (e.g. editors for strings) to complex (e.g. for setting colors in the Editor).

All changes made in custom property editors are applied throughout the environment immediately – for example, changing the background color of the Editor window changes not only any new Editor windows you open but also any that are currently open.

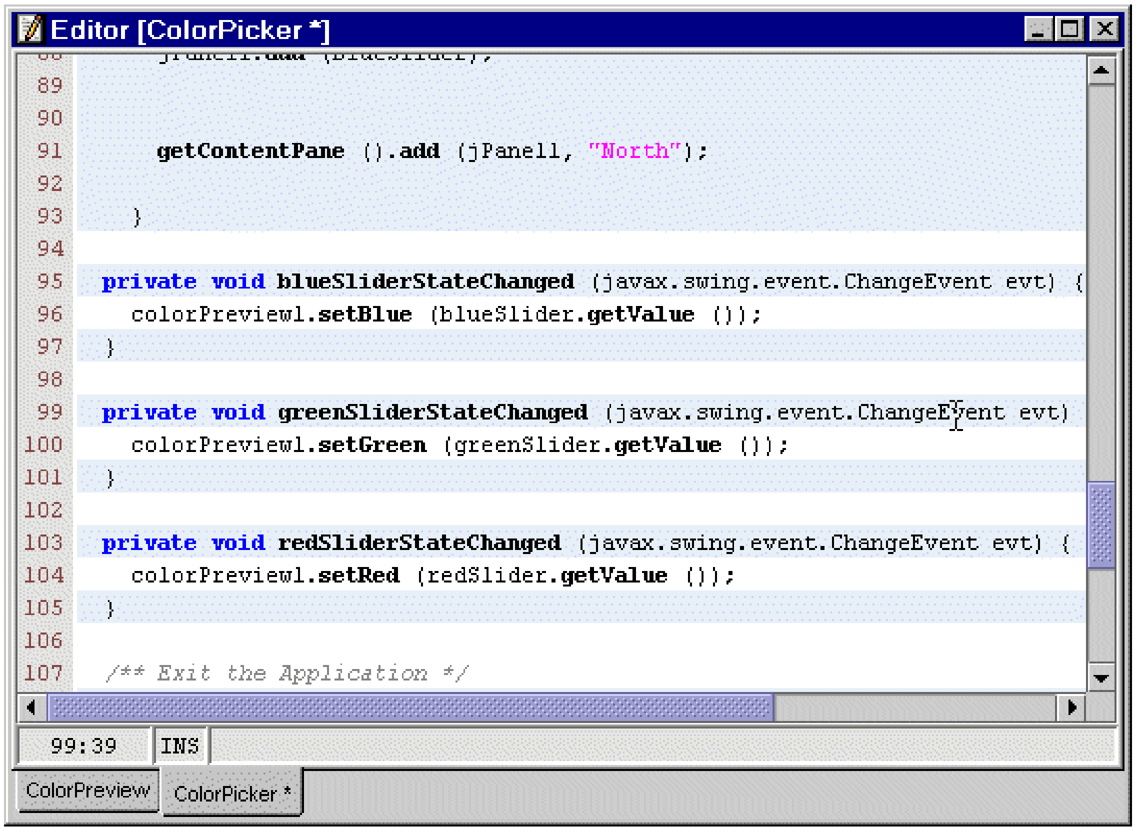
The Property Editor dialog box contains three buttons:

- The **OK** button closes the dialog and is shown only if the property is writable.
- The **Cancel** button reverts to the setting before the property editor was invoked.
- For properties that have default values, the **Default** button sets the property to its default value.

Editor

The Editor is a full-featured text editor that is integrated with the Form Editor, Explorer, Compiler, and Debugger. It is the default viewer for all Java, HTML, and plain text files as well as other types of files specified by modules (such as XML when the XML module is installed). Each of these open files is displayed in a multi-tab window – a single window with multiple tabs. Advanced features include customizable abbreviations and dynamic Java code completion. See “Editing Java sources” on page 41 for a description of how the Editor is integrated with other parts of the IDE.

The Editor window with Metal look and feel



```
        getContentPane ().add (jPanell, "North");

    }

  private void blueSliderStateChanged (javax.swing.event.ChangeEvent evt) {
    colorPreviewl.setBlue (blueSlider.getValue ());
  }

  private void greenSliderStateChanged (javax.swing.event.ChangeEvent evt)
    colorPreviewl.setGreen (greenSlider.getValue ());
  }

  private void redSliderStateChanged (javax.swing.event.ChangeEvent evt) {
    colorPreviewl.setRed (redSlider.getValue ());
  }

  /** Exit the Application */
```

Opening the Editor and navigating from tab to tab

Double-click a Java or text object in the Explorer to open the Editor. Any files that you subsequently open will also appear in the Editor with their own separate tabs in the bottom of the window. To flip between displayed files, simply click the tab of the file you want displayed. The tab of the currently visible file is highlighted. By right-clicking on a tab, you can bring up a context menu that gives you options to dock the tab to another window or clone the window. For more information, see "Multi-tab windows" on page 132.

A modified file the Editor window is marked with an asterisk (*) after its name on its tab. (The asterisk also appears after the file name in the window title.) If there are any unsaved modifications when the Editor is closed, a prompt will appear to save or discard changes, or cancel the operation.

Mouse and clipboard functions

The Editor uses standard mouse functions: click and drag the left mouse button to select a block of text, double-click the left button within a word to select that word, and triple-click to select an entire line. You can also select a block of text by clicking to place the caret where the block should begin, holding down SHIFT, and then clicking to determine the end of the selection.

You can select text and move it to and from the clipboard using the **Cut**, **Copy**, **Paste**, and **Delete** commands, which are available in the **Edit** menu, on the **Edit** toolbar, and in the popup menu (accessed by right-clicking on the selected text). You can use **Undo** to reverse the previous command and **Redo** to reverse the reversal. These commands are available in the **Edit** menu, on the **Edit** toolbar, in various context menus, and by using keyboard shortcuts (see below).

Editor keyboard shortcuts

A wide range of keyboard shortcuts are available in Forte for Java to speed the editing process, including:

- CTRL+c (or CTRL+INSERT) to copy
- CTRL+x (or SHIFT+DELETE) to cut
- CTRL+v (or SHIFT+INSERT) to paste
- CTRL+z to undo a command
- CTRL+y to re-do any command that has been undone
- CTRL+F3 – find word or selection marked by the caret
- CTRL+t – increase line or block indentation
- CTRL+d – decrease line or block indentation
- CTRL+b – go to the matching parenthesis, bracket, or brace
- CTRL+w – delete previous word
- CTRL+F2 – toggle bookmark (to bookmark a line – or remove a bookmark – in the Editor window)
- F2 – jump to the next bookmark in the document

For a complete list of shortcuts installed with Forte for Java, see “Editor Shortcut Keys” on page 162.

Word match

You may also may extend a series of letters to match with other text in the same document with the same prefix. After typing in a series of letters, press CTRL+k to find the previous instance of any word containing that prefix or CTRL+l to find any ensuing references. You may do this repeatedly to find multiple matches in the document. The IDE searches for a match not only in the current file, but also in all other opened files (in the order that they were last used).

Customizing Editor keyboard shortcuts

If the key combinations for the shortcuts do not suit you, you can change them.

To change the Editor's keyboard shortcut assignments:

1 Go to the `Editor Settings` node under the **Property Settings** tab in the Explorer.

2 Select the `Global Key Bindings` property and click on the **...** button. This will bring up the Key Bindings custom property editor.

3 In the custom property editor, edit the **Action** and **Key** fields to reassign the shortcut keys.

> **Note:**You cannot set a keyboard shortcut for anything that is not in the **Action** list in the custom property editor.

Editor abbreviations

To simplify editing of Java sources, a set of customizable abbreviations is built into the Editor, which can be used to expand a few pre-defined characters into a full word or a phrase. Defining abbreviations is useful for long and frequently-used Java keywords. For example, if you type `pu` and press SPACE, the text will be expanded to `public`. To enter characters without expansion, type SHIFT+SPACE – this enters a space without checking for abbreviations.

To change the Abbreviation table:

1 Go to `Property Settings / Editor Settings / Java Editor` in the Explorer and display its property sheet.

2 Select the `Abbreviations` property and click on the **...** button. This will bring up a custom property editor for abbreviations.

3 Select the abbreviation that you want to change in the list box and click **Edit**.

4 Another dialog will appear with a combo-box list for the abbreviations (**Action**) with the selected one showing and a field showing the selected expansion (**Key**). Change either or both values and click **OK** to make the changes or **Cancel** to go back to the Abbreviations dialog.

To add abbreviations:

◊ Follow the same steps as for changing abbreviations, but click **Add** in the Abbreviations dialog instead of **Edit**.

Find and replace

To find or replace text in a file open in the Editor window, press CTRL+f to invoke the Find dialog, or press CTRL+r to invoke the Replace dialog. The Find/Replace dialog gives you checkboxes which allow you to choose any combination of the following options:

- **Highlight Search** – to highlight all occurrences of the search text in the file

- **Incremental Search** – for the search engine to try to find the text as you type (without having to press the **Find** button)
- **Match Case** – to limit the search to text that has the same capitalization
- **Smart Case** – to limit the search to text that has the same capitalization when at least one character of the search text is upper case
- **Match Whole Words Only** – to match the search text only to whole words in the file
- **Backward Search** – to search in reverse order in the file
- **Wrap Search** – to continue the search at the beginning (or end) of the file

Java code completion

Forte for Java Community Edition 1.0 also has a dynamic code completion feature, where you can type a few characters and then bring up a list of possible classes, methods, variables, etc. that can be used to complete the expression.

To use Java code completion:

1 Type the first few characters of the expression (e.g. `import javax.` or `someFile.getP`).

2 Press CTRL+SPACE (or pause after entering a period). The code completion box will then appear.

3 Then use the most convenient combination of the following steps:

a. keep typing to narrow down the selection of items in the list.

b. use the navigation keys (UP arrow, DOWN arrow, PAGE-UP, PAGE-DOWN, HOME, and END) or the mouse to scroll through the list and select an expression

c. press ENTER to enter the selected method into your file and close the code completion box

d. press TAB to select the longest common substring matching the text you have typed (Bash-style context-sensitive completion) and keep the list box open

If you press ENTER for a method with parameters, replaceable text is given for the first parameter which you can then fill in. If the method takes multiple parameters, you can bring the list box back by typing a comma after you fill in the parameter.

Changing the key bindings for code completion

Just as you can with other Editor shortcuts, you can change the shortcut assignments for Java code completion.

To change the keyboard shortcut assignments for code completion:

1 Select the `Editor Settings / Java Editor` node under the **Property Settings** tab in the

Explorer.

2 On its property sheet, select the `Key Bindings` property and click on the **...** button. This will bring up the Key Bindings custom property editor.

3 In the custom property editor, edit the **Action** and **Key** fields to reassign the shortcut keys.

Adding your own classes to the code completion database

You can update the code completion database so that your own classes are among the choices offered when using the database.

To update the code completion database:

◊ In the Explorer or Object Browser, right-click the package containing the classes you want to add to the database and select **Tools | Update Parser Database...** from the context menu.

Form Editor

The Form Editor allows you to design applications visually. You can select items such as panels, scroll bars, menus, and buttons in the Component Palette and then place them directly on the Form Editor window. As you do, Forte for Java automatically generates the Java code to implement the design and behavior of the application. (The code is visible in the Editor window.) The Form Editor also uses your chosen Java layout manager, which controls the appearance of the components in a window, to control the layout in the Form Editor. If you choose a different layout manager or change its parameters, the Form Editor displays your changes immediately. For a comprehensive guide to using the Form Editor, see “Designing visually with the Form Editor” on page 84.

Debugger Window

The Debugger Window monitors breakpoints, watches and threads during a debugging session. Breakpoints provide the ability to break (pause) at certain points within the code during execution and examine the current state of the system. Watches can be used to monitor the values of variables used in the code during execution. You can also monitor all threads during execution, with access to a thread's call stack and local variables throughout the whole call stack.

Breakpoints, watches and threads are all presented on separate tabs in the Debugger window, along with a property sheet. In fact, this window is a view of the Debugger hierarchy presented in the Explorer, under `Runtime / Debugger`.

See “Debugging Java classes” on page 49 for a guide to using the Debugger.

Execution View

The Execution View provides a view of all applications currently being executed within the IDE. This window is actually a view of the Explorer hierarchy under `Runtime / Processes`.

By default, the Execution View is opened on the Running Workspace. When no applications are running, it simply displays `<No Processes Running>`. When an application or applet is running, it is listed by name. Each currently running process is listed. As for many objects in the IDE, a displayed process has a popup menu. In this window, the menu contains just one item: **Terminate Process**. This allows you to force termination of the selected process.

Output Window

The Output Window is a multi-tab window displaying output from any component that produces output – such as the compiler or executed application or applet. Output from each component is presented on a separate tab: the **Compiler** tab, **Debugger** tab, and a tab for each executed process labeled with the name of the application being run.

By default, the Output Window is visible on the Running Workspace and is automatically displayed when you compile (Editing Workspace), execute (Running Workspace), or debug (Debugging Workspace) an application.

The **Compiler** tab is visible after compiling a Java source and displays compilation output and standard error. The output is color coded: errors are marked red, other text is blue. Double-clicking an error line on this tab brings forward the Editor window displaying the source, highlights the incorrect line in red, and positions the caret at the exact location of the compilation error.

The Debugger tab is split vertically into two panes. The left pane displays the output of the application being debugged. The right pane displays useful debugging information such as details of the application's threads, thread groups and breakpoints as well as status messages from the debugger itself.

Any application currently being executed also has a separate tab in the Output Window. Its tab displays the standard output of the application. The standard input of the application (assuming the application tries to read anything) is also redirected here – more specifically, to the text field at the bottom of the window.

There are two properties which govern the use and re-use of application tabs: `Reuse Output Window Tab` and `Clear Output Window Tab`. These properties can be found on the property sheet of `Property Settings / Execution Settings` in the Explorer. If the `Reuse Output Window Tab` property is set to `True`, each individual application uses only a single output tab – that is, successive executions do not create new tabs. If the `Clear Output Window Tab` property is set to `True`, the tab is cleared before reuse. `Clear Output Window Tab` is useful only if `Reuse Output Window Tab` is set to `True`.

Context menus which provide window management options are available on the tabs – see "Window management" on page 130. You can also right-click in the body of the pane for the options **Copy to Clipboard** and **Clear Output**.

An application's output tab also provides a right-click popup menu, with the options **Terminate Process** and **Close Output Window Tab**.

Web Browser

Forte for Java Community Edition 1.0 includes the ICE Browser from ICEsoft, a built-in full-featured Web browser that is useful in both testing and providing easy access to online help sources. It enables standard browsing capabilities from within the IDE and is useful in reaching online help sources.

To open the Web Browser, do one of the following:

◊ Select **Web Browser** under the **View** menu on the Main Window.

◊ Press ALT+7, the default shortcut.

◊ Select a bookmark from **Help | Bookmarks** in the main menu.

◊ Open an HTML file or bookmark in the Repository.

Once the browser is open, it operates like any simple browser. To load a different page, type the URL in the **Location** field and press ENTER. The forward and back arrows cycle through previously-seen pages, the **Stop** icon stops the loading process, and the **Home** setting is set (by default) to the NetBeans website, http://www.netbeans.com/. Clicking the **History** icon gives you a list of all URLs you have loaded in the session and allows you to double-click on any of them to bring that web page back.

To open an additional Web Browser window:

1 Select **Window | Clone View** from the main menu to open a new browser window.

Another Web Browser window will open with the same Web page.

2 Type a new URL in the **Location** field and press ENTER to load a different Web page.

Note: You cannot open multiple Web Browser windows by other means. Selecting **Web Browser** or pressing ALT+7 causes the current window to revert to the default home page).

You may also want to set a different home page (the page that displays when you open the Web Browser or press the **Home** icon).

To set a different home page:

1 Go to `Project Settings / System Settings` in the Explorer.

2 Display the property sheet by selecting the **Toggle Property Sheet** icon or right-clicking on `System`

`Settings` and selecting **Properties** from the context menu.

3 Enter the new home page under the `Home Page` property and press ENTER.

Window management

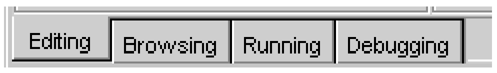
Forte for Java has the following features which, when combined, give you a great deal of flexibility in managing windows on your desktop:

- **Workspaces**, meaning window sets, each geared toward a given task (e.g. editing, browsing, running, or debugging) and which can be opened and closed as a group.
- **Multi-tab windows**, which hold multiple open files at once and allow you to flip between files by clicking on a tab.
- **Docking** of windows, which allows you to change the windows in which the files are displayed (for example, you can move a source file from a multi-tab Editor window to its own window or to a different already existing container window, such as the Output Window).
- **Cloning** of windows, which allows you to have the same file open in two different windows, making it easier to work on two different parts of the file at the same time.

Workspaces

As you progress through the development cycle (write, compile, debug, execute, edit, etc.) when working on any significant project, the screen can become cluttered with windows. While all of these windows are necessary for development, they are generally not all needed simultaneously.

Forte for Java workspaces enable you to efficiently manage a large number of windows and to group these windows into useful, logical sets. You can flip between workspaces by simply clicking the different workspace tabs on the Main Window.



Standard workspaces

The default workspace configuration is a standard and logical grouping of the most commonly used windows:

- **Editing** – Explorer; if any files are open, the Component Inspector, Form Editor window, and Editor window; after compilation of files, Output Window

- **Browsing** – Object Browser and Property Sheet window, and then the Editor window if any files are opened
- **Running** – Execution View, Output Window
- **Debugging** – Debugger Window, Output Window, and Editor (if any files are open)

Note: By default, the IDE automatically switches to the Running Workspace when you execute a program and to the Debugging Workspace when you start a debugging session.

Using workspaces

Being in a given workspace does not constrain what windows you can have open. You can open (e.g. from the **View** menu) or close any window without having to change the workspace. Whenever you open or close a window, the workspace reflects that change.

Upon exiting the IDE, the current state of each workspace is saved. This information includes open windows and their sizes and positions. When you next launch Forte for Java Community Edition 1.0, your workspaces will appear exactly as you left them.

Any given window can be open on any workspace—and can be open on more than one workspace simultaneously.

For example, to view the Editor window on both the Editing and Running workspaces:

1 Open the Editor window on the Editing Workspace.

2 Double-click on a Java object displayed in the Explorer.

The Editor window will open with that source.

3 Flip to the Running Workspace by clicking on the **Running** tab in the Main Window.

The first Editor window will no longer be visible, and you will see either the default set of Running Workspace windows or those you left open when last using this workspace.

4 Select the **Window** menu on the Main Window.

You will see a list of currently open windows, including **Editor[ ]** (with the name of the open source file in the brackets).

5 Select this item to open the Editor window on the current workspace – in this case, the Running Workspace.

The Editor will then be visible on both the Editing and Running workspaces.

For some types of windows, it is also possible to open separate instances of the same window, which act independently of each other. For example, if you have an Explorer window open on your Editing Workspace, you can open an entirely separate Explorer on the Debugging workspace by flipping to the Debugging Workspace and clicking the **Open Explorer** icon on the Main Window. This Explorer is a new window and does not display changes (concerning window size, nodes selected and expanded, etc.) made to an Explorer window on another workspace.

Workspaces are completely customizable: you may add, delete and rename the available workspaces. See "Customizing workspaces" on page 156 for further details.

Multi-tab windows

A multi-tab window presents multiple files as separate tabs in a single frame, enabling you to quickly and easily flip between these files, either by clicking on the tabs or by using the ALT+LEFT and ALT+RIGHT keyboard shortcuts.

Each tab has a popup menu of commands (**Save**, **Clone View**, **Close**, and **Dock Into...**), accessible by right-clicking on the tab itself. See below for more information on cloning, docking, and undocking windows.

Note: Multi-tab windows with only one file or component look like single windows – i.e. they don't have a tab. Thus commands that appear on context menus for tabs must be accessed elsewhere (such as from the Main Window) when in a single window. Once a second file is opened in or docked to the window, tabs for both items appear.

Undocking and docking windows

While the multi-tab window enables quick and efficient access to more than one file or view, only one of those files is visible at any one time. At times it is useful to look at different files or views simultaneously, side by side. Forte for Java makes this possible with its **docking** feature, which allows you to dock (move objects into multi-tab windows) and undock (move object views to single-view windows).

Undocking

Any object open in a multi-tab window (with at least two objects open) can be undocked and presented in its own stand-alone, independent window. In this way you can simultaneously view separate sources, side by side. Undocked windows may also be docked back to the "parent" multi-tab window (See Docking below.)

To undock the active tab of the active window:

◊ Select **Undock Window** from either

- the **Window** menu on the Main Window; or
- the tab's context menu (available by right-clicking on the tab);

◊ Or Select **Dock into...| Single Frame** from either

- the **Window** menu on the Main Window; or
- the tab's context menu (available by right-clicking on the tab).

The active object will be removed from its original window and will appear in a new window. This new window may be repositioned, resized, moved to another workspace, and even closed – all independently of the parent window. You can position the windows side-by-side to view sections of source simultaneously or copy and paste code between windows.

Docking

Docking windows allows you to reduce clutter on your desktop without having to close any objects and to view objects in a window that is more convenient for you.

You can dock any window tab into any named container window such as:

- the Output Window
- the Debugger Window
- the Editor window

or

- other windows such as the Debugger Window, Object Browser, and Form Editor window when they are already open

or

- a new unnamed multi-tab window

Almost any window can be docked into one of the windows above. This includes any Editor tab, the Component Inspector, the whole Explorer window, the Property Sheet, the Object Browser, the Web Browser, the Execution View, the Debugger tabs, and the Output Window. Frames from multi-tab windows can also be docked into new unnamed single-view frames.

To dock an undocked window into a different window:

1. Click on the window to be docked to activate it.
2. Select **Dock Into...** from the **Window** menu on the Main Window and then one of the windows given in the submenu.

   The undocked window will close, and the object will open as a new tab in a multi-tab window.

To dock a tab in a multi-window into a different window:

1. Select **Dock Into...** from either

   - the **Window** menu on the Main Window (making sure that the undocked window is the active window); or

- the tab's context menu (available by right-clicking on the tab).

2 Select one of the window choices in the **Dock Into...** submenu.

Note: Separate Explorer tabs cannot be docked to other windows, but the whole Explorer can. If you dock the Explorer into a multi-tab window, there will be two sets of tabs at the bottom of the window when the Explorer is active in that window (the top set is for the main Explorer nodes and the bottom set is for the Explorer as a whole and the other objects in the window).

Cloning windows

It is often useful to view separate sections of the same file simultaneously. Forte for Java has a **clone** function to accomplish this. You can clone the view of any Editor file or the Web Browser.

To clone a window:

1 Make sure the file to be viewed in the cloned window is on the active tab.

2 Select **Clone View** from either

- the **Window** menu of the Main Window; or
- the tab's popup menu.

A new tab will open in the same multi-tab window, displaying the same content.

3 Undock this new view of the file or dock it into a different window.

For more information on docking, see "Undocking and docking windows" on page 132.

Modules in Forte for Java

Forte for Java's modular architecture means that all parts – even those central to the functionality of the IDE, like the Editor, Debugger, and Form Editor – are in module form. There are also base modules for things such as the HTTP server, the Web Browser, and creation of JavaBeans. Modules are also available for advanced features such as Database Explorer, Servlets/JSP, RMI, CORBA, etc. In addition, you can create your own modules or add third-party modules.

Managing modules

All modules installed in the IDE have their own node in the Explorer under `Global Settings / Modules`. Their property sheets contain the `Enabled` property which you can use to activate or deactivate the module. The other five properties (including two on the **Expert** tab) are read-only and give basic information about the module.

Adding modules

Modules can be easily added and updated with the Auto Update feature.

To update or add a module to the IDE using Auto Update:

◊ Select **Help | Update Center...** from the main menu.

The IDE will access the NetBeans website (and partner websites) where the latest versions of all modules are available and determine which modules are missing or not current in your IDE. It then presents you with a list of these modules and allows you to select the ones you would like to download. After you have made your selection, the modules are downloaded and installed into the IDE.

You can also add modules manually by right-clicking on the `Modules` node and selecting **New Module from File**. The Open dialog will appear allowing you to browse for the JAR file containing the module. If you select a valid module and click **OK**, the module will be installed into the IDE.

Uninstalling modules

To uninstall a module:

1. Under the **Global Settings** tab in the Explorer, expand the `Modules` node and select the module you want to delete.
2. On the property sheet, select the `Enable` property and set it to `False`.

The next time you start Forte for Java Community Edition 1.0, the JAR file for that module will not be loaded into the IDE.

Should you wish to later reinstall the module, you can repeat the same procedure, but switch the `Enable` property to `True`.

Exploring objects

The Explorer is not only a place where you can manage files, but also a place where you can create objects and manage their properties as well as control IDE settings. Virtually anything that can be done in Forte for Java can be done through the Explorer using:

- the context menu (available for each node by right-clicking on the node)

- the property sheet (accessible from the context menu or by pressing the **Toggle Property Sheet** icon in the Explorer's toolbar)

There are four top-level folders in the Explorer's hierarchy:

- `Repository` – contains work objects.
- `Runtime` – contains a list of currently running processes and debugger information.
- `Project Settings` – holds all settings (such as Repository, compilation, execution, debugging, and workspace settings) specific to a Forte for Java project. If you install the Projects module, it is possible to have multiple projects and thus multiple sets of Project Settings.
- `Global Settings` – holds all general IDE settings for menus, toolbars, modules, etc.

Repository

The Repository is the most important segment of the IDE's Explorer. It holds all files that the IDE uses and all files you create. The Repository gives a unified view of files of all different types. You can access objects from different sources by mounting different file systems and JAR and ZIP archives in the Repository.

File systems and the class path

The concept of file systems is critical to the architecture of the IDE.

A file system represents a hierarchy of files and folders. File systems can represent plain local files (or the network drive, depending on the operating system) or ZIP or JAR archives. Once mounted, the file system transparently hides any differences between these sources.

Some default items are added to the Repository when the IDE is launched. User file systems can be viewed in the Explorer under `Repository`. However, there are additional file systems used by the IDE which are mounted at startup and hidden. All mounted file systems, including hidden ones, are visible in the Explorer under `Project Settings / Repository settings`. Using the context menus and their property sheets, you can hide, unhide, or reorder them, control their behavior, etc.

Tip: Once a new file system has been mounted, it is equivalent to having added the archive or directory to the class path accessible by the IDE. The mounted packages and classes are immediately available, and can be edited, compiled, run, etc.

Mounting file systems correctly

An important issue to note when mounting new file systems is the point at which you mount them.

In the same way that the class path must contain the correct hierarchy of directories to access the required packages and classes, the point at which you mount those packages and classes is very important. For example, mounting a file system from the point `C:\work` will not give the IDE correct access to the package `project1` stored as `c:\work\myprojects\project1`. In this case, this file system should be mounted at `C:\work\myprojects`, so that `project1` is the top-level package.

To mount a new file system

◊ Right-click on the `Repository` node in the Explorer and choose **Add Directory...** from the popup menu; or

◊ Select **Tools | Add Directory** from the main menu.

A dialog will appear which allow you to choose the directory to mount.

To mount a new JAR or ZIP archive

◊ Follow the same procedure as for adding directories, except choose **Add JAR** instead of **Add Directory**.

Note: Mounted JAR archives are read-only.

To mount a single file

◊ Select **Open File** from the first toolbar or the **File** menu in the Main Window. A file chooser will appear which will allow you to browse your file system and select a file, which you can then add to the Repository. See "Adding a file to the IDE" on page 40.

To remove mounted directories or JAR or ZIP files from the Repository:

◊ Right-click on the item and select **Remove From Repository**.

Order of file systems

The order of file systems in the Repository is also significant. If files with the same name and hierarchy (e.g. `com/foo/Foo.java`) exist in two different mounted directories, the first will be loaded during execution or debugging, even if you have selected the second one.

To change the order of file systems:

◊ Right-click on the node of the file system under `Project Settings / Repository Settings` and select **Move Up** or **Move Down** from the context menu; or

◊ Right-click on the `Repository Settings` node and select **Change Order** from the context menu.

Working with packages

Objects stored in the Repository are organized in **packages** which correspond to file folders. All development packages you create when writing an application in Forte for Java are stored in the Repository, where they can be added, removed, and edited.

When starting a new development project, identify the data path or file system (see above) to use for your work, right-click on that path, and select **New Package** from the popup menu. Once you enter a name for the new package, it appears in the Explorer under the path you have selected. You can create several layers of packages (meaning packages within packages) in the same way. Deleting these packages is simple – either choose **Delete** from the package's popup menu or press the DELETE key on your keyboard.

Note: When you cut or copy source files and paste them to a different package, the sources' package declarations are automatically updated to reflect the new package. If you copy and paste a file to the same package, the pasted copy is automatically given a unique name (which you can change if you wish, either from the context menu, or by clicking on it in the Explorer to select it and then clicking again for an in-place rename.

The popup menu commands in the Repository allow a wide range of operations – from creating new packages to changing properties. The following a is list of menu items that appear in various context menus.

Table 6: Package Popup Menu Commands

Command	***Description***
Explore from Here	Opens a new Explorer window with the selected package as the root.
Refresh Folder	Updates the view, reflecting any changes to files in the folder made outside the IDE.
Compile	Compiles all uncompiled or modified objects in the selected package, at the selected level in the hierarchy.
Compile All	Compiles all uncompiled or modified objects in the selected package and recursively in all sub-packages.
Build	Compiles or re-compiles all objects (whether already compiled or not) in the selected package at the selected level in the hierarchy.
Build All	Builds all objects in the selected package and recursively in all sub-packages.
Cut / **Copy** / **Delete** / **Rename**	In addition to the standard keyboard commands, Forte for Java enables cutting, copying, pasting, deleting, and renaming from the popup context menu.
Paste | Copy	Pastes a copy of the object most recently copied under the selected node.

Command	*Description*
Paste | Create Link	Creates a link under the selected node to the most recently copied object. The object remains stored in the location where it was copied, but it can also be opened from the node where the link is pasted.
Paste | Instantiate	Creates a new instance of the copied template (only available when a template is on the clipboard).
Paste | Serialize	Serializes the instance of the copied JavaBeans object and places it in the selected package.
Paste | Default instance	Places the default instance of the copied JavaBeans object in the selected package, meaning that the name of the class is stored and the class name is provided as the default constructor in the pasted copy.
New Package	Creates a new, empty package as a sub-folder of the selected package.
New from Template	Creates a new object in this package, using one of the pre-built templates available under the **Templates** node in the Explorer.
Tools | Update Parser Database	Updates the Java completion database with the classes of the selected package, thus making those classes available in addition to the standard SDK classes when using the Java code completion feature in the Editor.
Properties	Opens a separate Property Sheet window showing properties of the selected object(s).

Working with objects

While working in Forte for Java Community Edition 1.0, you operate with objects rather than plain files. Each object represents one or more files on disk. Each object is shown with its own icon and properties. The object types used in the IDE are:

Table 7: Object Types

Icon	*Object Type*
📁	**Package** – A package (folder)—on disk or in a JAR or ZIP archive.
📄	**Java object** – Represents one Java source file (`.java`). Its children represent methods, variables, constructors, and inner classes acquired by parsing the Java source.

Icon	*Object Type*
	Form object – represents one Java source file which can be edited visually in the Forte for Java Form Editor. The two types of subnodes are: 1) classes with methods, variables, constructors, and inner classes acquired from parsing Java source; and 2) items representing components on the form (visual hierarchy).
	Class object – represents one Java class without source code. Children are methods, variables, constructors, and inner classes acquired from Java reflection.
	Serialized prototypes – files with a `.ser` extension, which are serialized objects.
	HTML object – represents an HTML file.
	Text object – represents a text file.
	Image object – represents GIF or JPEG images. You can view these with the built-in image viewer.

Three advanced operations can be done with JavaBeans and serialized prototypes. You can:

- Customize them (using the **Customize Bean** command) and make serialized prototypes from the customized object.
- Copy and paste them into the Component Palette.
- Copy and paste them directly into the Component Inspector (without the need to install them in the Component Palette first).

There are several ways of creating new objects including selecting the **New From Template** command in:

- the Main Window toolbar
- the **File** menu
- a Repository package's context menu

See "Creating new objects from templates" on page 148 for more information.

You can also copy objects from one package and paste them in another using **Copy** and **Paste** from the context menus. There are special paste options for JavaBeans objects.

Objects can be removed using the **Delete** icon in the Explorer toolbar or by pressing DELETE on the keyboard.

For each object type, you can access its popup menu which includes:

Table 8: Common Object Commands

Command	*Description*
Open	Opens the default viewer for the object type – usually the Editor window. Also opens up the Form Editor window and Component Inspector for visual classes.
View	Opens an HTML object in the default applet viewer.
Compile	Compiles selected object(s).
Execute	Runs the selected object.
Cut / **Copy** / **Paste** / **Delete** / **Rename**	Standard clipboard-style operations.
New | Method (or Constructor, Initializer, Variable, Inner Class, Inner Interface)	Creates a new element (of the type selected in the submenu) in the selected class or source file. These commands are available on the popup menu for the relevant category node (e.g. `Fields`)
New | Property	Invokes the New Property Pattern dialog for creating a new JavaBeans property for the selected bean.
Tools | Create Group	Create a group (object composed of links to one or more files, allowing you to access them from the same place in the IDE). See “Group” on page 150.
Tools | Add to Component Palette	Add selected object to the Component Palette.
Tools | Synchronize	Force synchronization of the selected source file with the interfaces it implements.
Save as Template	Publishes the selected object as a template for future use.
Properties	Opens a property sheet showing properties of selected object(s).

Runtime

The second major grouping in the Explorer hierarchy is `Runtime`. Settings under the `Runtime` node display runtime information for execution (executed processes) and debugging (breakpoints, thread groups, and watches), as well as any external services as provided by extension modules and their connection to the IDE.

Processes

The `Processes` node lists all processes currently being executed within the IDE. The popup menu for each of these items contains just one element: **Terminate Process**. This allows you to force termination of a process.

This view is mirrored by the Execution View.

Debugger

During a debugging session, the Debugger shows individual watches, threads and breakpoints. By right-clicking on Breakpoints or Watches, you can also add a new breakpoint or watch.

There's a detailed description of the Debugger in “Debugging Java classes” on page 49.

Project Settings

This main node comprises all of the IDE system settings that can be set for a project. These include Repository settings, internal and external compiler configuration, Debugger settings, window settings, external browser or applet viewer configuration, and much more. You can view and edit the various options for the selected Project Settings item on its property sheet. As in the Repository, context menu items such as **Customize Bean**, **Copy**, and **Paste** are available for many of the Property Settings nodes.

Changes made to Project Settings are automatically saved when you exit the IDE. To save changes without closing the IDE, select **File | Save Settings** from the main menu.

Note: The Projects module allows you to group files into projects and to have a different set of Project Settings for each project. The Save Settings command is thus replaced by the **Save Project** command in the **Project** menu. Otherwise, Project Settings apply universally.

Below are descriptions of each item in Project Settings.

Repository Settings

`Repository Settings` lists all mounted file systems. You can customize the appearance and behavior of file systems that are currently mounted – local directories as well as JAR and ZIP archives. If a new file system is added in the Repository, it automatically appears as one of the items in the `Repository Settings` tree. You can use the context menu on the `Repository Settings` to add new file systems and re-order current file systems. See "Repository" on page 136 for more information.

On the property sheet, you can hide file systems or make them read-only. Under the **Capabilities** tab on the property sheet, you can disable the IDE from performing certain tasks on the file system (searching for sources to compile, searching for classes to execute, searching for classes and sources to debug and searching for HTML pages with documentation). These properties are of particular interest if you want to mount the Java 2 SDK itself and debug it. You can set the `compile` property on the SDK file system to `False` to prevent the IDE from trying to compile it.

Tip: Capabilities settings apply to entire file systems. However, if you would like to keep individual sources in a file system from being compiled, executed, or debugged, you can change the source's `Compiler`, `Executor`, or `Debugger` property (under the **Execution** tab of its property sheet) to `(do not ...)` to accomplish this.

Debugger Settings

The Debugger Settings property sheet contains various settings allowing you to choose whether to automatically have classes compiled before debugging, whether the Editor should track the current position of the Debugger, which workspace to use when debugging, etc. The **Expert** tab contains various path settings and the `Classic` setting (which should be set to `True` if HotSpot is installed.

Editor Settings

The Editor Settings property sheet gives you broad control over the behavior of the Editor and appearance of files displayed in the Editor window. You can set key bindings, make abbreviation lists, use the color chooser to customize foreground and background colors for different types of text, choose caret type, etc. Under the `Editor Options` node are three subnodes, representing types of files that can be edited in the Editor: `HTML Editor`, `Plain Editor`, and `Java Editor`. These three subnodes each have separate property sheets, allowing you to make distinct customizations for each type of Editor. For a description of all of these settings, see "Editor Settings reference" on page 182.

Execution Settings

The Execution Settings property sheet gives you the option of clearing the Output Window when executing, having separate Output Window tabs for each execution, and automatically compiling source before execution. You can also choose the workspace used when running applications.

Form Objects

The Form Objects settings affect the appearance of the Form Editor window during design time, code generation settings, etc.

HTTP Server

The HTTP Server property sheet controls the built-in HTTP (Hypertext Transfer Protocol or "Web") server. Among other things, you can establish the host, determine access to the server, and change the port.

Java Elements

These settings control the display names of element nodes in the Explorer. You can enter a combination of plain text and substitution codes. For example, if you enter: `class {n}` in the `Classes` property, a class called `MyClass` will be represented as `class MyClass` in the Explorer hierarchy. See "Java Elements settings reference" on page 187 for a complete list of the substitution codes.

Java Sources

The Java Objects settings allow you to enable or disable external compilation, set the source parser delay, and set a strings table.

The strings table is a list of substitution keys for templates. When the key appears in a template (marked by an underscore on each side of the substitution key in the template), the string assigned to the key appears in place of the key in any objects created from that template.

For example, the `USER` substitution key is assigned your user name. If a template has the key `_USER_` placed anywhere in the text, your user name will appear in place of `_USER_` in any object created from this template.

The `Java Sources` node also contains the `Source Synchronization` subnode where you can enable or disable the source synchronization feature as well as set its return mode.

Object Browser

The sole property for this node allows you to set custom package filters for use in the Object Browser. (You can also do this from the Object Browser window).

Open File Server

The property sheet for this node allows you to configure the server that listens to open requests if you use the IDE as an external viewer or editor. See "Opening files from external processes" on page 157.

Output Window

These settings control the colors and fonts used in the Output Window.

Print Settings

These settings control the appearance of files printed from the Editor. From the `Print Settings` node, you can modify header, footer, line wrapping, and line ascent correction properties for printouts of all Editor files. On the property sheets for each of the three subnodes (`Java Editor`, `Plain Editor`, and `HTML Editor`), you can set line numbering and specify detailed background and foreground coloring for different types of text within the files.

Property Sheet

Here you can set the appearance and behavior of the Property Sheet.

System Settings

On the System Settings property sheet, you can set the look and feel of the IDE, choose whether or not to have the IDE prompt you to confirm deletions, and select whether to show tips on startup. There are also settings for proxy port and proxy server, which you can set if you need to use a proxy server for HTTP or FTP connections with the Web Browser and Update Center. If you want to use a proxy, you must also set the `Use Proxy` property to `True`.

Workspaces

Under `Workspaces` are nodes for every workspace. Under these nodes are subnodes showing the windows in each workspace. Since the configuration of each workspace is updated whenever you open or close a window within that workspace, these subnodes also change to reflect the windows that are currently part of that workspace. You can also configure workspaces and set up new ones in the Explorer. See "Customizing workspaces" on page 156 for more information.

Compiler Types

Here you can set specific configurations for each type of compiler type (the actual system command that is invoked when you compile an object, which includes the path to Java, the working directory, and any arguments). In Forte for Java, you can use the default compiler types of each category (e.g. internal compilation and external compilation), modify the default compiler types, or write new ones. You can then associate any of your classes to one of these compiler types if the default compiler type assigned for that class does not suit your needs.

You can also access the custom property editor for compiler types from the property sheet of individual classes. See “Switching compilers” on page 44 and “Adding and modifying service types” on page 72 for more information.

Executor Types

Here you can set specific configurations for each type of “executor” (the actual system command that is invoked when you execute an object, which includes the path to Java, the working directory, and any arguments). You can use the default executors of each category (external, internal, and applet execution), modify the default executors, or write new ones. You can then associate any of your classes to one of these executors if the default executor assigned for that class does not suit your needs.

There is a node for each type of executor (such as `External` for stand-alone applications, `ThreadExecutor` for internal execution, `Applet Execution`, and others for modules such as RMI, JSP, etc.) and under these nodes are listed the executors themselves (first executor for each type is labelled `Default`).

You can also access the custom property editor for executors from the property sheet of individual classes. See “Configuring external executors” on page 47 and “Adding and modifying service types” on page 72 for more information.

For Applet Execution, you can modify the `External Viewer` property. It is not possible to modify the Thread (internal) executors.

Debugger Types

Just as with Compiler Types and Executor Types, you can make custom configurations for debugger types (the actual system command invoked when you debug an object, including the path to Java, the working directory, and any arguments). You can use the default debugger types of each category (e.g. standard debugging or JPDA debugging), modify the default debugger types, or write new ones. You can then associate any of your classes to one of these debugger types if the default debugger type assigned for that class does not suit your needs.

You can also access the custom property editor for debugger types from the property sheet of individual classes. See “Setting the debugger” on page 54 and “Adding and modifying service types” on page 72 for more information.

Global Settings

The Global Settings govern the overall look, content, and performance of the IDE, regardless of the project you are working on.

Actions

The action pool under `Actions` in the Explorer hierarchy stores all actions (commands) that are available in the IDE, most of which (but not all) are already represented in the menus and toolbars. Since you cannot delete actions from the action pool, this gives you the freedom to remove whatever items you want from the menus and toolbars without permanently removing them from the IDE. You can add actions to the main menu and toolbars by copying them from the action pool and pasting them to `Global Settings / Menu` or `Global Settings / Toolbars`.

Menu

Under `Menu`, you will find various customization settings for the menus on the Main Window and all of their elements. Under the `Menu` node, you will find settings for each menu on the Main Window menu bar, including `File`, `Edit`, `Build`, `Debug`, `Tools`, `Window`, and `Help`. Under these are subnodes for the menu items and any submenus. You can add, remove, and customize commands for each menu and menu item. You can also change the order of the menu items, enable or disable them, and insert separators.

Toolbars

Under `Toolbars`, you will find various customization settings for the buttons of the toolbars on the Main Window and all of their elements. You can add, remove, and customize commands for each toolbar button. You can also change the order of the toolbar items, enable or disable them, and insert separators.

Startup

Here you can insert classes that the IDE will run at startup. Classes that set general keyboard shortcuts are stored here. Internal execution is usually required for startup classes.

Templates

Here you will find the standard set of templates, in several categories: `AWTForms`, `Classes`, `Dialogs`, `SwingForms` and `Other`. Each category contains several templates for creating new objects. See “Using templates” on page 148. You can also create your own categories and templates – see “Creating your own templates” on page 150 – and modify existing templates (including their property sheets).

Object Types

Under this node, you can change the order of the object types (loaders) and customize their properties. Each loader recognizes specific types of files (usually by their extension) and groups them with similar files as they are loaded. This affects where new objects that can fall under multiple object types are placed in the IDE (and ultimately how they are handled) since they are automatically dropped into the first category that they match with when they are imported into the IDE.

Important: In some cases, the order is crucial: e.g. forms objects must come before Java source objects, which must come before classes objects. Only experts should modify these settings.

For some object types, such as text, image, HTML, and class objects, you can set the extensions that the loader recognizes (under `Extensions` on the object type's property sheet).

Modules

This node lists all of the modules currently installed in the IDE. The property sheets shows their name, version numbers, and whether they are enabled.

Component Palette

In the Component Palette section of the Explorer, you can add, remove, move, change the order of, or edit components. From a component's popup menu, you can move it up or down, cut, copy, delete, or set properties. Some items have an `Is Container` property, which, when set to `True`, means that other components can be added to that component in the Form Editor.

Bookmarks

This node holds all of the bookmarks (URLs) that appear under **Help | Bookmarks** in the main menu. Using the context menus, you can add, remove, move, change the order of, or edit bookmarks, as well as create sub-folders to organize them.

Using templates

Templates are a powerful IDE tool. You can use them to create new objects, which then can be used as a basis for creating more complex objects. Java components such as Swing and AWT containers are one type of template. There are also various templates for classes, dialogs, and other objects such as HTML files, text files, and bookmarks. Even “empty ” classes have templates. You can also create your own templates.

Creating new objects from templates

You can create a new object from a template in one of the following ways.

To create a new object from a template from the Main Window:

1. Choose **New from Template** from either the toolbar or the **File** menu. The Template dialog with tabs for different groupings of templates will appear. The default categories are **AWTForms**, **SwingForms**, **Classes**, **Dialogs**, and **Other**.
2. Click on a tab to select the category, click on an icon of one the individual templates in that category, and then click **OK** (or double-click the icon).
3. The Instantiate Template will appear on the screen with a combo box containing a list of all selectable file systems in the `Repository` and a tree of all packages in the root directory (default package) of the selected file system. Select the file system from the combo box and then the package where you want the object to be created, fill in the **Object Name** field, and click **OK**.

To create a new object from a template from the Repository or Object Browser

1. In the Explorer's `Repository` or the Object Browser, select the package where you want to create the new object, open its popup menu, and select a template under **New from Template**.
2. In the Instantiate Template dialog that appears, enter the name of the new object and click **OK**.

It is also possible to create new objects by double-clicking on the template itself, either under `Global Settings / Templates` or under `Repository` (if the object has been set as a template there).

You can also create an object from a template by copying the template (using the **Copy** command from the **Edit** menu or toolbar, context menu, or keyboard shortcut) and then selecting **Paste | Instantiate**.

After the new object is created, the Editor will open for this object if the object supports editing. (For example, no editor is available for Class objects.)

Other templates

These templates are all found under the **Other** category.

Package

A package for storing objects. Creating an object from this template is equivalent to selecting the **New Package** command.

Bookmark

Selecting this template automatically creates a bookmark with the URL http://www.netbeans.com/ and opens up that web page in the default Web browser.

To edit the URL in a bookmark:

◊ In the Explorer, right-click on the bookmark and select **Edit URL...** in the popup menu.

A dialog will appear prompting you to enter the new URL.

Group

A group is a cluster of aliases for files somehow related to each other. It is particularly useful if you want to simultaneously work on files located in separate parts of the IDE. The template itself is "empty". Once you create an object from the template, you can "fill" the object by pasting links from it to other files. You can also save such a group as a new template.

To add a file to a group:

1 In the Explorer, select an object you would like to add to the group and select **Copy** from the context menu.

2 Right-click on the group's node and select **Paste** from the context menu.

HTML

An HTML file.

Text

A plain text file.

Creating your own templates

In Forte for Java Community Edition 1.0, any object can be used as a template. There are two ways to set an object as a template:

To set an object as a template:

1 Select the desired object in `Repository` (under Explorer) and select **Save as Template** from the popup menu.

2 In the Save as Template dialog that appears, select the template category where you want the new template placed.

Or:

1 Select the desired object in `Repository` and change its **Template** property to `True` in the property sheet. You can also copy this object to a template category under the `Templates` node; this will make the object visible in the window that appears when the **New** command is chosen on the menu or toolbar.

2 If you want the new template visible in the menus and dialogs used for creating objects from templates, copy it to a template category under the `Global Settings / Templates` node in Explorer.

The difference between a template and a normal object is only in what is done when you double-click on it in the Explorer. For a plain object, a default Editor window is opened (if available); on templates, a dialog appears allowing you to create a new object from the template.

If you want to open the Editor on a template, choose **Edit** from the template's popup menu.

Modifying existing templates

You can also modify existing templates. This might be particular desirable if you generally work with non-default service types (for compilers, executors, and debuggers). You can modify the content of a template in the Editor and the properties of a template on its property sheet.

To modify the contents of a template:

◊ Right-click on the template in the Explorer and select **Open** in the Explorer. (The IDE's set of default templates can be found under the `Templates` node of the **Global Settings** tab, though it is also possible to store templates in the Repository.)

To modify the properties of a template:

1 Select the template in the Explorer (usually under the `Templates` node under **Global Settings** – see above.

2 Open its property sheet and make the changes you desire.

Tip: If you want to make the same change to multiple templates (e.g. switch all of them to JPDA debugging), you hold down the CTRL key and select multiple template source files. Any changes you then make on the property sheet will apply to all of the selected templates.

Customizing the environment

Customizing menus and toolbars

You can organize all menu and toolbar commands in the IDE to your preferences so that you have quick and easy access to them. From a customization point of view there are only minor differences between the menu and toolbars, because they are both used as generic containers to visually display a list of commands.

Changing commands in menus and toolbars

You can add and delete existing commands on the menus and toolbars using clipboard-style operations.

To add a new command to a menu or toolbar:

1 Select the command in the `Global Settings / Actions` hierarchy in the Explorer and copy it (using **Copy** from the **Edit** menu or toolbar, the node's context menu, or CTRL+c).

2 Under `Global Settings / Menu` (or `Global Settings / Toolbars`), select the subnode for the menu or toolbar under which you would like to place the command.

3 Select **Paste | Copy** from either the **Edit** menu or context menu.

To move a command to a different menu or toolbar:

1 Select the command you want to move in the `Global Settings / Menu` or `Global Settings / Toolbars` hierarchy in the Explorer and cut it (using **Cut** from the **Edit** menu or toolbar, the node's context menu, or CTRL+x).

2 Under `Global Settings / Menu` (or `Global Settings / Toolbars`), select the subnode for the menu or toolbar under which you would like to place the command.

3 Select **Paste** from either the **Edit** menu, the **Edit** toolbar, or context menu (or use the CTRL+v keyboard shortcut).

Note: If you delete a command supplied with Forte for Java from the default menus or toolbars, it is not completely deleted from the IDE. You can restore a command to a menu or toolbar by copying it from `Global Settings / Actions` (where all available commands are stored) and pasting it to a menu (listed under `Global Settings / Menu`) or toolbar `Global Settings / Toolbars`).

To group related menu commands with a separator:

1 Right-click on the desired parent container under `Global Settings / Menu` in the Explorer and select the **New | Menu Separator** command from the context menu.

2 Enter a name for the separator in the dialog that appears and click **OK**.

3 The menu separator will appear as the last item in the menu.

To group related toolbar commands with a separator:

1 Copy one of the separators in the list of menu items under one of the subnodes of `Global Settings / Toolbars`.

2 Right-click on a toolbar node under `Global Settings / Toolbars` and select **Paste | Copy** from the context menu.

3 The menu separator will appear as the last item in the menu.

To change the order of commands within their parent container:

◊ Select the command you want to move, and move it using the **Move Up** or **Move Down** command in the popup menu; or

◊ Select the parent container and choose **Change Order** from its popup menu. A dialog will appear which will allow you to rearrange the commands by dragging and dropping with the mouse or by using the **Move Up** and **Move Down** buttons on them.

Tip: Each menu item may also have a keyboard accelerator – a letter or digit underlined in the item itself which enables you to select that item by pressing the appropriate key on the keyboard instead of dragging the mouse to it. This letter or digit is marked by & in the **Name** property of the item.

Creating new menus and toolbars

You can also create new menus and toolbars in the IDE.

To create a new menu:

1 Right-click on the `Global Settings / Menu` node (or on the `Global Settings / Toolbars` node if you are adding a toolbar), and select **New | Menu** from the context menu.

2 In the New Menu dialog that appears, type the name of the new menu and click **OK**.

To create a new toolbar:

1 Right-click on the `Global Settings / Toolbars` node and select **New | Toolbar** from the context menu.

2 In the New Toolbar dialog that appears, type the name of the new menu and click **OK**.

Dragging toolbars

The Main Window toolbars can be managed with mouse operations and context menus directly in the Main Window.

To move a single toolbar:

◊ Click on the “handle” on the left side of the toolbar (marked by two light-colored vertical bars) and drag it with the mouse.

To move consecutive toolbars simultaneously:

◊ Right-click on the “handle” of the left-most of the toolbars and drag it with the mouse. That toolbar and all toolbars to its right will be moved.

To switch toolbar rows:

◊ Hold down the CTRL key, click on the “handle” of a toolbar and drag the toolbar up or down. All toolbars in that row will move in the direction you are dragging (trading rows with the toolbars previously in the row you have dragged to).

Toolbar context menu and toolbar configurations

There is also a context menu on the Main Window which you can use to manage toolbars. You can access this context menu by right-clicking on any empty space in the toolbar area of the Main Window. The context menu is divided into three parts:

- a list of available toolbars
- a list of toolbar configurations
- the **Save Configuration** command

The available toolbars are listed with checkboxes. By checking and unchecking these items, you can control which toolbars are displayed in the Main Window.

The toolbar configurations are shown with radio buttons, allowing you to check the configuration (group of toolbars and their positions) you want. By default, there are two configurations. The **Standard** configuration includes the **Debug** toolbar, which contains only a selection of debugging commands. The **Debugging** has the **DebugFull** toolbar with all Debugger commands. By default, the Running and Debugging workspaces both use the **Debug** toolbar.

The **Save Configuration** command allows you to save the current configuration of toolbars and create a name for it (thus creating a new configuration).

You can set the toolbar configuration separately for each workspace.

To set a toolbar configuration for a workspace:

1. Select the node for the workspace under `Project Settings / Workspaces` in the Explorer.
2. Open up the workspace's property sheet by right-clicking and selecting **Properties** from the context menu or selecting the **Toggle Property Sheet** icon on the Explorer toolbar.
3. For the `Toolbar config` property, select the configuration from the drop-down list.

Customizing shortcuts

Keyboard shortcuts offer an alternative method of performing operations for users who prefer the keyboard to the mouse. These are also useful for frequently-used commands, when a single keystroke is much faster and more efficient.

In Forte for Java, shortcuts are stored in the `Keys` class run at startup. You can find and edit this class in the Explorer under `Global Settings / Startup`.

Customizing the Component Palette

You can also customize the Component Palette. Under `Global Settings / Component Palette` you can find the current Component Palette categories. By expanding a category node, you can view a list of all components in that category.

To create a Palette category

1 Right-click on the `Component Palette` node and select **New Palette Category** from the popup menu.

2 Enter the new category name in the dialog and click **OK**.

After clicking, the new category will appear as the last item in the Explorer listing, and a tab for the new category will appear in the Component Palette.

Standard clipboard operations are available at each level of the Explorer's `Component Palette` hierarchy. Individual components can be cut, copied, and pasted between component categories. For example, you can copy the components you use most to a new category for quick and easy access. You can also cut, copy and paste component categories.

Component categories can be renamed—either by selecting **Rename** on the popup menu or using in-place renaming in the Explorer.

It is also possible to customize the ordering of component categories and the components within a category.

To move a component category

◊ Right-click on the category node and select **Move Up** or **Move Down** from the context menu; or

◊ Right-click on the `Component Palette` node and choose **Change Order** from the context menu to invoke the Customize Order dialog, which lists all of the categories, has **Move Up** and **Move Down** buttons, and also allows you to re-order items by drag-and-drop.

To change the order of a component within a category

◊ Right-click on the node for the component and select **Move Up** or **Move Down** from the context menu; or

◊ Right-click on the node of the component’s category and choose **Change Order** from the context menu to invoke the Customize Order dialog.

To remove a component or component category from the Component Palette

◊ Right-click on the node for the component or the category and select **Delete** from the context menu; or

For information on adding JavaBeans to the Component Palette, see “Adding JavaBeans to the IDE” on page 113.

Customizing workspaces

Workspaces are entirely customizable via the `Workspaces` node under `Project Settings` in the Explorer. Expand `Workspaces` to see a list of current workspaces. These workspace nodes can in turn be expanded to see a list of currently open windows on each workspace.

Each level of this hierarchy can be cut, copied and pasted, using commands on either the popup menu or the **Edit** menu of the Main Window. Existing workspaces can be renamed, again via the popup menu, the workspace's property sheet, or using in-place editing of the name in the Explorer.

Workspace clipboard operations act recursively: copying a workspace copies all windows present on that workspace. If you then paste the new workspace to the workspace hierarchy, these windows are copied to that new workspace.

To create a new workspace:

1 Right-click on the root `Workspaces` node under `Project Settings` and select **New Workspace** from the popup menu.

  A dialog will open allowing you to specify the name of the new workspace.

2 Type your name and click **OK**.

  The new item will appear in the given tree in the Explorer and immediately be displayed on the Main Window.

If you need to expand the workspace tab area on the Main Window to see your new workspace, click and drag the workspace tab divider to the right. If there are more workspaces than are visible in the workspace tab area, left and right scroll buttons will be displayed allowing you to scroll through all available workspaces.

To add windows to the new workspace, you can either:

◊ Select the workspace by clicking on its tab in the Main Window and then open up the windows that you want to be on that workspace from the **Windows** menu or toolbar; or

◊ Under `Global Settings / Workspaces` in the Explorer, copy windows from other workspace nodes and paste them to the new workspace.

Note: Only the first option works if you want to add a window not already in a workspace to the new workspace.

You can also customize which workspace the IDE automatically switches to when you start debugging or start execution.

To set the workspace the IDE switches to for debugging:

1 Under `Project Settings` in the Explorer, right-click on the `Debugger` node and select **Properties**.

2 One of the properties is `Workspace` - this determines which workspace the IDE will

automatically switch to upon initiating a debugging session. Click on the value of this property (by default set to `Debugging`) and select a workspace from the pull-down list of all current workspaces. You can also set this property to `None` if you don't want the workspace to change.

Similarly for execution, you can select which desktop the IDE will switch to when running an application by setting the `Workspace` property of the `Project Settings / Execution` node.

For more information on workspaces, see "Workspaces" on page 130.

Opening files from external processes

Many programs have the ability to associate specific commands with file types (generally by extension or MIME type), so that the user may select a program to view or edit that type of file. Examples include the Microsoft Windows desktop / Explorer; assorted file browsers available for Unix and other operating systems, as well as alternatives on Windows; web browsers; version control systems; mail programs; etc. If you wish, you can associate the IDE with certain file types, such as Java sources or property bundles, so that the IDE's editor will be opened when you double-click on the file.

Using the remote launcher manually

To make this work, first try opening a file in the IDE using the remote launcher manually. Open a command prompt appropriate for your system (e.g. MS-DOS console or Unix shell). Find the JAR file for the Open File module – probably included in your Forte for Java installation. Now decide on a file to open and type something like this:

```
C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
     Files\forte4j\modules\openfile.jar"
 "C:\My Development\com\mycom\Foo.java"
```

If the IDE is running, you should see `Foo.java` displayed in the Editor, or the usual mount dialog may appear. See "Adding a file to the IDE" on page 40. Please specify complete paths to all files to make sure the same command will work later. The quotes are desirable, especially on Windows, in case you have any directories with spaces in their names.

Associating the launcher with a type of file

Now you are ready to associate the Open File launcher with some file type in your application. The details of how to do this will of course vary depending on the system. Generally, you must specify everything before the filename in the above example command as the "external editor".

As an example, to associate the IDE with `*.java` files in Windows:

1 Open any Windows Explorer window and select **View | Options**. Go to the **File Types** tab.

2. See if there is some kind of entry for Java source files (the name may vary, depending on what software has been installed) – the entry should specify the file extension `java`. Click **New Type...** if necessary; otherwise select the entry and click **Edit....**
3. In the **Actions** box, you need at least one action which opens the file in the IDE – you can name it whatever you like, usually `Open`. Click **New...** or **Edit...** as appropriate.
4. In the resulting action dialog, supply whatever name you like, and give as the application the command you typed before, but replacing the filename with `%1`:

   ```
   C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
        Files\forte4j\modules\openfile.jar" "%1"
   ```

5. Normally you will also click **Set Default** (it will be bold if it is the default) to make it the default action for double-clicks.
6. If desired, you may similarly associate other file types – recommended are `*.properties`, `*.ser`, `*.form`, `*.class`, `*.jar`, `*.zip`, etc.

   Now you should just be able to double-click any of these files in Windows and the IDE will open them for you. For other systems such as Linux, if you use a file browser (such as from KDE or Gnome, or a separate browser) you can similarly set up file associations.

Customizing file opening

The Open File launcher has several options which can be used to customize the process of opening files remotely. You may pass these on the command line to the launcher, for example:

```
C:\jdk1.2.2\jre\bin\javaw -jar "C:\Program
     Files\forte4j\modules\openfile.jar" -port 2121 "C:\My
     Development\com\mycom\Foo.java"
```

Note that you may also pass multiple files at once to the launcher. The available options are:

`-host`*`hostname-or-IP`* – select an alternate machine to open files on. The specified machine should be the one running the IDE. You may want to also turn on `-nocanon` (below). The Open File server must be configured to accept requests from the machine running the command (below).

`-port` *`port-number`* – Select an alternate port to send the Open File requests to (via UDP). By default, port 7318 is used. The Open File server must be configured to listen on that port (below).

- `-canon` (default) or `-nocanon` – when using `-canon`, any relative file names passed to the launcher are first converted to an absolute name. If using `-host` to open files remotely, you may want to use `-nocanon` and always specify the correct absolute file name for the machine running the IDE.

- `-wait` and `-nowait` (default) – by default, the launcher sends a message to the IDE asking it to open a file, and then immediately exits as soon as the IDE has opened it. (The command will return a success code (zero on Unix) normally, or an error code (nonzero on Unix) if there was a problem opening the file.)

Some applications demand that an editor be used to make a specific set of changes to a file and then finish with it. If this is the case, specify `-wait` to the launcher. For example: many version control systems on Unix provide the option to edit a change log message with an external editor, or to clean up the results of a merge before continuing with a check-in. If you use such a system, and set the Open File launcher with `-wait` as your external editor, then the file will open in the IDE as usual, but the launcher will wait to exit. The launcher will not exit until you have made some changes to the file and saved them. Currently it is not sufficient to just close the Editor window – you must actually save changes. If you decide you do not want to make any changes, simply type a space (for example), delete it, and then ask to save (to let the launcher know you are finished with the file).

- `-help` – Display a brief usage message.

Open File Server properties

When using the IDE as an external viewer or editor, you can control the behavior of the server that listens to open requests. In the Explorer, go to `Project Settings / Open File Server` and open the property sheet. You can configure how the server should behave by adjusting the following properties:

- **Access Restriction** – By default, only open requests from the same machine are honored, so that other people cannot open files in your IDE. If you set this property to `Any Host`, any incoming requests will be honored. (Many local area networks prohibit arbitrary traffic from entering the network from the Internet, so this is not necessarily as unsafe as it sounds. Also note that the request is sent via UDP, which most firewalls will not pass.)
- **Port** – You may configure the port the server runs on, which might be necessary if another program is using port 7318.
- **Running** – You may set this property to `False` to stop the server, or `True` to turn it back on.

Stopping the server only affects using the launcher – it does not affect opening files from within the IDE using the toolbar button or menu item.

Appendix A

Default Keyboard Shortcuts

Table 9: Naming Convention for Function Keys

Name	Used For
LEFT, UP, RIGHT, DOWN	Cursor arrow keys
BACKSPACE, SHIFT, ENTER, DELETE	These keys correspond to common special keys on most keyboards.
END, HOME, PgUp, PgDown	Other cursor repositioning keys
F1, F2, ... F12	Numbered function keys (across the top of the keyboard)

Modifier keys

Most keyboard shortcuts use one or more modifier keys such as ALT, CTRL, META, and SHIFT, which you must hold down while pressing the other key in the combination.

Note: These names are conventional, but some systems (especially the X Window System) may use different names. For example, META is often equivalent to ALT.

Global Shortcut Keys

Context-Sensitive Help

F1 displays help for the selected component in the GUI, such as a window, a pane or a tab in a window, dialog, Explorer node, etc.

Clipboard

Table 10: Clipboard shortcuts

Press	*To*
CTRL+c	Copy current selection into the clipboard
CTRL+x	Cut current selection into the clipboard
CTRL+v	Paste current content of the clipboard
DELETE	Delete selection

Note: In some cases, it is better to use these shortcuts from popup menus in the Explorer, because they give further options. For example, when copying a file from one directory to another, you have the option of creating a duplicate file or creating a link to the original.

Form Editor Shortcut Keys

These shortcuts are useful when you are designing visual forms using the Form Editor.

Table 11: Form Editor shortcuts

Press	*To*
CTRL+F10	Switch to Form Editor window
CTRL+F11	Switch to Editor window
CTRL+F12	Switch to Component Inspector
CTRL+SHIFT+F10	Toggle Test Mode (form specific)
CTRL+SHIFT+F11	Toggle Design Mode (form specific)
CTRL+SHIFT+F12	Toggle Grid (global option)

Editor Shortcut Keys

Cursor movements

Table 12: Cursor shortcuts

Keyboard shortcut	With no selected text	With selected text
RIGHT	Move cursor one character to the right	Deselect text and move cursor one character to the right
LEFT	Move cursor one character to the left	Deselect text and move cursor one character to the left
DOWN	Move cursor to the next line	Deselect text and move cursor to the next line
UP	Move cursor to the previous line	Deselect text and move cursor to the previous line
SHIFT+RIGHT	Create selection and extend it one character to the right	Extend selection one character to the right
SHIFT+LEFT	Create selection and extend it one character to the left	Extend selection one character to the left
SHIFT+DOWN	Create selection and extend it to the next line	Extend selection to the next line
SHIFT+UP	Create selection and extend it to the previous line	Extend selection to the previous line
CTRL+RIGHT	Move cursor one word to the right	Deselect text and move cursor one word to the right
CTRL+LEFT	Move cursor one word to the left	Deselect text and move cursor one word to the left
CTRL+SHIFT+RIGHT	Create selection and extend it one word to the right	Extend selection one word to the right
CTRL+SHIFT+LEFT	Create selection and extend it one word to the left	Extend selection one word to the left
PgDown	Move cursor one page down	Deselect text and move cursor one page down

Keyboard shortcut	With no selected text	With selected text
PgUp	Move cursor one page up	Deselect text and move cursor one page up
SHIFT+PgDown	Create selection and extend it to the beginning of the next page	Extend selection one page down
SHIFT+PgUp	Create selection and extend it to the beginning of the previous page	Extend selection one page up
HOME	Move cursor to the beginning of line	Deselect text and move cursor to beginning of line
END	Move cursor to the end of line	Deselect text and move cursor to end of line
SHIFT+HOME	Create selection and extend it to the beginning of the previous page	Extend selection to beginning of line
SHIFT+END	Create selection and extend it to the beginning of line	Extend selection to end of line
CTRL+HOME	Create selection and extend it to the beginning of end of line	Deselect text and move cursor to beginning of document
CTRL+END	Move cursor to the end of document	Deselect text and move cursor to end of document
CTRL+SHIFT+HOME	Create selection and extend it to the beginning of end document	Extend selection to beginning of document
CTRL+SHIFT+END	Create selection and extend it to the beginning of end of document	Extend selection to end of document
CTRL+a	Select whole document	Select whole document
CTRL+e	Remove current line	Remove current line

Shifting text right/left

Table 13: Shortcuts for horizontally shifting text

Keyboard shortcut	With no selected text	With selected text
TAB	Insert tab	Shift selection right
SHIFT+TAB		Shift selection left
CTRL+t	Shift line right	Shift selection right
CTRL+d	Shift line left	Shift selection left

Find shortcuts

Table 14: Find shortcuts

Keyboard shortcut	With no selected text	With selected text
CTRL+f	Show Find dialog	Show Find dialog and show selected text as the text to find
F3	Search for next occurrence	
SHIFT+F3	Search for previous occurrence	
CTRL+F3	Search for occurrence of the word that the cursor is on	Search for occurrence of the selected text
ALT+SHIFT+h	Toggle highlight search	
CTRL+g	Show Goto Line dialog	

Other shortcuts

Table 15: Miscellaneous shortcuts

Press	To
CTRL+w	Remove the word before the cursor
F2	Go to next bookmark
CTRL+F2	Toggle bookmark

Press	To
CTRL+k	Word Match - find previous matching word
CTRL+l	Word Match - find next matching word
CTRL+b	Find matching bracket
CTRL+SHIFT+b	Select block between current bracket and matching one
SHIFT+SPACE	Insert space without expanding abbreviation
ALT+g	Go to variable declaration
ALT+u then g	Prefix the identifier with `get`
ALT+u then s	Prefix the identifier with `set`
ALT+u then i	Prefix the identifier with `is`
ALT+o	Opens the source based on where the cursor is.
ALT+h	Scroll the text up so that the cursor moves to the top of the window while remaining at the same point in the text.
ALT+m	Scroll the text so that the cursor moves to the middle of the window while remaining at the same point in the text.
ALT+l	Scroll the text down so that the cursor moves to the bottom of the window while remaining at the same point in the text.
SHIFT+ALT+h	Move the cursor to the top of the window.
SHIFT+ALT+m	Move the cursor to the middle of the window.
SHIFT+ALT+l	Move the cursor to the bottom of the window.
ALT+p	Go to previous entry in the jump list.
ALT+n	Go to next entry in the jump list.
SHIFT+ALT+N	Go to the first jump list entry in the next component
SHIFT+ALT+P	Go to the last jump list entry in the previous component
ALT+J	Select the identifier the cursor is on (or deselect the selected identifier).

Special Java Shortcut

Table 16: Java code completion

Press	To
CTRL+ENTER	Open Java Completion list box with matching entries. (TAB completes the word and leaves the list box open. ENTER completes the word and closes the list box.)

Explorer Shortcut Keys

The following items are associated with the Explorer (though the Explorer does not have to be open to use all of them.

Table 17: Explorer shortcuts

Press	*To*
CTRL+n	Create New Object
CTRL+o	Open Explorer or activate currently open Explorer window
CTRL+s	Save
CTRL+SHIFT+s	Save All (unsaved documents)
Delete	Delete (selected node)

Window Shortcut Keys

The following are shortcuts for opening or activating specific windows and other window-related functions.

Table 18: Window shortcuts

Press	*To*
ALT+0	Explore From Here
ALT+1	Open the selected object’s property sheet in a separate window
ALT+2	Open/activate the Explorer window (same as CTRL+o)

Press	***To***
ALT+3	Open/activate the Editor window (works only if a text or source file is already open)
ALT+4	Open/activate the Output Window
ALT+6	Open/activate the Execution View
ALT+7	Open/activate the Web Browser
ALT+8	Open/activate the Component Inspector
CTRL+F4	Close the active window (or tab in multi-tab window)
ALT+SHIFT+LEFT	Switch to previous workspace
ALT+SHIFT+RIGHT	Switch to next workspace
ALT+LEFT	Switch to previous tab (in Editor window)
ALT+RIGHT	Switch to next tab (in Editor window)
CTRL+u	Undock window (undocks currently activated tab in multi-tab window into a single window)
CTRL+p	Print (file in active Editor window)
CTRL+F1	Do Javadoc index search (if Javadoc module is installed)

Build Shortcut Keys

These shortcuts keys mirror commands available in the **Build** menu.

Table 19: Compile/Build shortcuts

Press	***To***
F9	Compile all out-of-date files under selected node
ALT+F9	Build (compile all files) under selected node
ALT+c	Stop Compilation
ALT+F7	Go to Previous Error Line
ALT+F8	Go to Next Error Line
CTRL+F9	Run

Debugger Shortcut Keys

These shortcut keys mirror commands available in the **Debug** menu.

Table 20: Debugger shortcuts

Press	*To*
F5	Go (Start Debugging)
SHIFT+F5	Finish Debugging
CTRL+F8	Toggle Breakpoint
SHIFT+F8	Add Watch
F7	Trace Into
F8	Trace Over
CTRL+F7	Step Out

Appendix B

Default Java Editor Abbreviations

Java Editor abbreviations are defined in the custom `Abbreviations`, under `Project Settings / Editor Settings / Java Editor`, which you can edit to add, delete, and change shortcuts. The following table lists the default Java Editor abbreviations.

Table 21: Java Editor abbreviations

Abbreviation	*Expands To*
sout	System.out.println ("
serr	System.err.println ("
impa	import java.awt.
impb	import java.beans.
impi	import org.openide.
impj	import java.
imps	import javax.swing.

Abbreviation	***Expands To***
impS	import com.sun.java.swing.
impq	import javax.sql
psf	private static final
psfi	private static final int
psfs	private static final String
psfb	private static final boolean
Psf	public static final
Psfi	public static final int
Psfs	public static final String
Psfb	public static final boolean
ab	abstract
bo	boolean
br	break
ca	catch (
cl	class
cn	continue
de	default:
el	else
ex	extends
fa	false
fi	final
fl	float
fy	finally
im	implements
ir	import
iof	instanceof
ie	interface
nu	null

Abbreviation	Expands To
pr	private
pe	protected
pu	public
re	return
sh	short
st	static
sw	switch (
sy	synchronized
tr	transient
th	throws
tw	throw
twn	throw new
twni	throw new InternalError();
twne	throw new Error()
vo	void
wh	while
En	Enumeration
Ex	Exception
Gr	Graphics
Ob	Object");
Re	Rectangle
St	String
Ve	Vector
pst	printStackTrace();
tds	Thread.dumpStack();

Appendix C

Main Window Menus

Menus

The following sections provide lists and short descriptions each of the entries on the Main Window menus.

File Menu

- **New From Template** – create a new object. The Templates folder will open, allowing a predefined template to be used as the basis for the new object. Available template categories are: AWT Forms, Classes, Dialogs, Other, and Swing Forms. See "Using templates" on page 148 for more information on templates.

- **Open Explorer** – open a new Explorer window.

- **Open File** – open a file that is currently in the IDE, or mount a file system to the IDE and open a file in it.

- **Object Browser** – open the Object Browser window.
- **Save** – save the current object. If there is more than one object currently open in the multi-tab Editor window, the source in the currently selected tab is saved. If there is more than one Editor window open (that is, one or more sources have been undocked), the object in the currently selected Editor is saved. Save is disabled when no modified objects are currently open or the selected object is unmodified.
- **Save All** – save all current unsaved open objects.
- **Save Settings** – saves all of the current settings. (Settings are saved automatically when exiting the IDE.)
- **Print** – print the file active in the Editor or selected in the Explorer.
- **Exit** – exit the IDE. You will be prompted to selectively save any currently open unsaved objects, discard their changes, or cancel. Workspace configurations and other settings changes are saved.

Edit Menu

- **Undo** – undo last action.
- **Redo** – redo last action.
- **Cut** – cut selected object or text to clipboard.
- **Copy** – copy selected object or text to clipboard.
- **Paste** – paste contents of clipboard.
- **Delete** – delete selected object.
- **Find ...** – find in text.
- **Replace ...** – replace in text.
- **Goto ...** – goto line number in Editor window.

View Menu

- **Explore from Here** – open a new Explorer, with the selected node as the root.
- **Properties** – make the Property Sheet window for the selected object the active window (or open the property sheet).

- **Explorer Window** – make the currently open Explorer the active window (or open it if it is not already open).
- **Editor Window** – make the currently open Editor the active window (or open it if it is not already open).
- **Output Window** – make the currently open Output Window the active window (or open it if it is not already open).
- **Debugger Window** – make the currently open Debugger Window the active window (or open it if it is not already open).
- **Execution Window** – make the currently open Execution Window the active window (or open it if it is not already open).
- **WebBrowser** – make the currently open Web Browser window the active window (or open it if it is not already open).
- **Component Inspector** – make the currently open Component Inspector window the active window (or open it if it is not already open).
- **Look and feel** – list of all available look and feels as a submenu with the current one indicated by a check mark. Selecting one in the submenu switches the whole IDE's look and feel.
- **Workspaces** – list of all available workspaces as a submenu with the current indicated by a check mark. Selecting one in the submenu switches the current workspace.

Build Menu

- **Compile** – compile selected object. If a folder is selected, compile all sources in that folder that are uncompiled or have been modified since the last compilation.
- **Compile All** – recursively compile (compile all uncompiled or out-of-date files in a folder and all its sub-folders).
- **Build** – force compilation of all classes, whether current or not.
- **Build All** – recursively build; build a folder and all its subfolders.
- **Clean** – delete all `.class` files in the selected package
- **Clean All** – recursively delete all `.class` files in the selected package and its sub-packages
- **Stop Compilation** – stop the current compilation process.
- **Next Error** – jump to the next error in the Output Window.

- **Previous Error** – jump to the previous error in the Output Window.
- **Set Arguments** – set command line arguments to be passed to an executed application. (May also be set in the `Arguments` property under the **Execution** tab of the property sheet of the class).
- **Execute** – execute the selected object.

Debug Menu

- **Go** – initiate a debugging session and run the program.
- **Connect** – connect the Debugger to an already running process.
- **Finish Debugger** – end the current debugging session.
- **Suspend All** – Suspend all threads in the Debugger.
- **Resume All** – Resume debugging of all threads in the Debugger.
- **Trace Into** – trace into the method the debugger has halted at.
- **Trace Over** – trace over the method the debugger has halted at.
- **Step Out** – halt execution after the current method finishes and control passes to the caller.
- **Toggle Breakpoint** – toggle a breakpoint on or off at the current line. The line will be highlighted blue when a breakpoint is set.
- **Add Breakpoint** – add a breakpoint. A dialog will appear prompting you to type where it should it be set.
- **Add Watch** – watch a variable.

Tools Menu

- **Add Directory** – mount a new directory under the Repository.
- **Add JAR** – mount a new JAR archive as a file system under the Repository.
- **Remove From Repository** – unmount the selected file system from the Repository.
- **Install New JavaBean** – install a new JavaBean to the Component Palette.
- **Tools | Update Parser Database...** – update the Java completion database with the classes of the selected package, thus making those classes available in addition to the standard SDK classes

when using the Java code completion feature in the Editor.

Window Menu

- **Clone View** – clone the current window. (opens a second view of the current window as a new tab on the multi-tab window, or in a separate window if the original window is not tabbed).
- **Undock Window** – undock the current window from the parent multi-tab window. Opens in a completely separate and independent multi-tab window.
- **Dock Into...** – dock selected undocked window to one of the choices in the submenu (Output, Debugger, single, or new multi-tab window).
- **Next Tab** – flip to the next tab in the multi-tab window.
- **Previous Tab** – flip to the previous tab in the multi-tab window.
- Currently opened windows – all windows currently open in any workspace are listed at the bottom of the **Window** menu. Selecting one of these activates that window, or opens it in the current workspace if it is not already there.

Help Menu

- **Browse Online User's Guide** – opens the HTML copy of the User's Guide in the Web Browser.
- **NetBeans Home on the Web** – open the browser and connect to the NetBeans website at http://www.netbeans.com/ If you are using a machine which connects to the internet via a dial-up modem, this may mean your modem attempts to dial and connect to your ISP. This depends on your local system configuration.
- **Features**– display a submenu of modules with separate documentation. If you select a module in the submenu, user's documentation will be opened in the Web Browser.
- **Bookmarks**– display a submenu with bookmarks. If you select a bookmark, the web page connected with it will open in the Web Browser.
- **Tip of the Day...** – display the Tip of the Day dialog which opens by default when you run Forte for Java.
- **Update Center** – connect with the NetBeans website to automatically install new or updated modules to your IDE.
- **About** – display the Forte for Java Community Edition 1.0 About dialog.

Toolbars

There are toolbar items for many of the menu commands as well. You can identify the commands for each toolbar item by holding the mouse over the icon to invoke its tool tip label.

Appendix D

Reference Guide to Project Settings

Repository Settings reference

The following settings are available for each file system under the `Repository Settings` node (the last four appear under the **Capabilities** tab).

Table 22: File System properties

Property	***Description***
Hidden	If True, the file system is not displayed in the Repository
Read Only	If True, you may not modify any files in the directory.
Root Directory	The root directory of the file system (should be the top of the package hierarchy.
Valid	If True, the file system is valid and usable.
Compile	If True, files in the directory can be compiled.
Debug	If True, files in the directory can be debugged.
Doc	If True, files in the directory.
Execute	If True, file in the directory can be executed.

Compiler types reference

Table 23: External Java Compiler service properties

Property	***Description***
Debug	If `True`, the compiler produces code with debug information. Must be turned on to debug with source tracking.
Deprecation	If `True`, the compiler treats use of deprecated methods as errors.
Encoding	If `True`, character encoding is used in Java sources.
Error Expression	A regular expression in POSIX format describing the format of the error output of the specified compiler. You can also define a custom expression, if using a different compiler, and save that setting. Select one of the pre-defined sets for error descriptors: `Sun javac`, `Microsoft jvc`, or `IBM jikes + E`.
External Compiler	Path to executable compiler. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Optimize	If `True`, the compiler optimizes generated bytecode.
Debug tag replace	String used in command line as debug (e.g. `-g`)

Property	Description
Deprecation tag replace	String used in command line as deprecation (e.g. `-deprecation`)
Optimize tag replace	String used in command line as optimize (e.g. `-O`)

Table 24: Internal Java Compiler service properties

Property	Description
Debug	If `True`, the compiler produces code with debug information. Must be turned on to debug with source tracking.
Deprecation	If `True`, the compiler treats use of deprecated methods as errors.
Encoding	If `True`, character encoding is used in Java sources.
Optimize	If `True`, the compiler optimizes generated bytecode.

See http://java.sun.com/products/jdk/1.2/docs/tooldocs/solaris/javac.html for more information on Javac.

Executor types reference

Table 25: External Execution service properties

Property	Description
External Executor	Path to Java launcher. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Identifying Name	The name for the executor type by which the classes reference it.
Boot class path	Boot class path on startup.
Class path	System class path on startup.
Library path	List of modules and special libraries the IDE uses.
Repository path	Read-only value filled on startup.

Table 26: Applet Execution service properties

Property	*Description*
External viewer	Applet viewer. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Name	The name for the applet executor type by which the classes reference it.

Debugger types reference

Table 27: Applet Debugging and Default Debugging properties

Property	*Description*
Classic	Set to True if HotSpot is installed.
Debugger	Path to the debugger. You can use the custom property editor to define the process and arguments, using a set of substitution codes.
Name	The name for the debugger type by which the classes reference it.
Boot class path	Boot class path on startup.
Class path	System class path on startup.
Library path	List of modules and special libraries the IDE uses.
Repository path	Read-only value filled on startup.

Debugger Settings reference

Table 28: Debugger properties

Property	*Description*
Followed by Editor	If true, the current line (the line on which the debugger is currently stopped) is visible during debugging (default=true).
Run Compilation	Compile sources before debugging.
Show Messages	If true, additional messages about the debugger are printed to the output window. (default=true)
Version	Version name of the installed debugger.
Workspace	The name of the workspace to which the IDE switches when the debugger starts, or None if the workspace should not be switched.

Editor Settings reference

Global Editor settings

This setting applies to editing of all types of files (e.g. Java, HTML, plain text) in the Editor.

Table 29: Global Editor Settings property

Property	*Description*
Global Key Bindings	Allows you to use the custom property editor for key bindings to set the key combinations for the available shortcuts

Editor settings by type of editor

Most Editor Settings are divided by type of editor – Plain, HTML, Java and Properties editors in Forte for Java Community Edition 1.0 (with other types added by extension modules). Even though the different editors have many of the same properties in common, most of the properties for the different types of editor are configured separately. The only exception is Global Key Bindings which is configured globally for all types of editors.

Table 30: Expert properties (available separately for each editor type)

Property	Description
Caret Blink Rate	Rate in milliseconds that the text caret blinks.
Insert Caret	The type of caret that appears when in insert mode. From the drop-down list you can choose between line, thin line, and block.
Insert Caret Color	Choose a color by clicking on the property and then either selecting a color from the drop-down list or invoking the custom property editor for colors by pressing **...**.
Italic Insert Caret	If True, caret is italic when in insert mode.
Italic Overwrite Caret	If True, caret is italic when in overwrite mode.
Line Height Correction	Multiplier to adjust height of lines.
Line Number Margin	Invokes custom property editor to set up placement of line numbers.
Margin	Invokes custom property editor to set top, bottom, left, and right margins.
Overwrite caret	The type of caret that appears when in overwrite mode. From the drop-down list you can choose between line, thin line, and block.
Overwrite caret color	Caret color when in overwrite mode.
Scroll Find Insets	Specify how much space should be reserved on each side of text located with the Find command
Scroll Jump Insets	Specify for each four directions how much the view should jump when the scrolling goes off of the screen
Status Bar Caret Delay	The delay in milliseconds between the time the caret stops moving and its position is updated in the status bar
Status Bar Visible	If True, the status bar (which shows information such as current line number, whether the Editor is in insert or overwrite mode, etc.) is displayed at the bottom of the window.

Table 31: Standard properties (available separately for each editor type)

Property	Description
Abbreviations	Allows you to use the Abbreviations (Map) custom property editor to set abbreviations that the Editor will automatically expand to longer strings when you type them.

Property	***Description***
Expand tabs to spaces	If True, tabs in the document are converted to spaces padded to the same column.
Font Size	If you change this property, the font for all tokens will be set to that size, regardless of sizes set in the `Fonts and Colors` property.
Fonts and Colors	Provides access to a custom property editor where background and foreground colors as well as fonts can be set for various syntax tokens.
Key Bindings	Clicking on the property and then clicking on the **...** button invokes the Key Bindings property editor, which allows you to add, remove, and edit key bindings.
Line Numbers	If True, the lines are numbered.
Number of Spaces per Tab	Number of spaces a block is indented when you enter TAB.
Tab size	Number of spaces between each tab stop in the editor. The tab stops are used when importing files into the editor.

Table 32: Java-only Editor Settings

Property	***Description***
Add Space before Curly Bracket	If True, a space is added before curly bracket (i.e {) in generated code.
Auto Popup of Java Completion	If True, the Java code completion box automatically appears when appropriate.
Compound Bracket on Next Line	If True, compound brackets generated automatically are put on the line following the previous code
Delay of Java Auto Completion Popup	Delay in milliseconds before the Java Completion popup appears.

Text types available for coloring

Using the custom color editor for the `Fonts and Colors` property for the `HTML Editor`, `Plain Editor`, and `Java Editor` nodes, you can provide separate background and foreground coloring for the following types of text in these three types of files:

- **HTML Editor –** argument, block comment, bookmark, default, error, guarded, highlight-search, incremental-search, int, line number, operator, text selection, status bar, status bar – bold, string, tag, and text.

- **Java Editor –** block comment, bookmark, char, default, error, float, guarded, hex, highlight-search, identifier, incremental-search, int, keyword, line comment, line number, long number, method, octal number, operator, text selection, status bar, status bar – bold, string, and text.
- **Plain Editor –** bookmark, default, error, guarded, highlight-search, incremental-search, line number, text selection, status bar, status bar – bold, and text.

Execution Settings reference

Table 33: Execution properties

Property	*Description*
Clear Output Tab	If `True`, output is cleared from the Output Window before reuse
Reuse Output Tab	If `False`, the IDE creates new tabs in the Output Window for each execution
Run Compilation	If `True`, the IDE first compiles before executing a file
Workspace	The window to activate when executing a file (Running, Browsing, Editing, Debugging, or None)

Form Objects reference

The last six properties on this table are listed under the **Expert** tab on the property sheet.

Table 34: Form Objects properties

Property	*Description*
Event Variable Name	The name of the variable generated in the signature of the event handler method for the event object. For example, "`evt`" is the variable name in "`private void button1ActionPerformed (java.awt.event.Action-Event evt)`".
Generate Null Layout	If True, forms using the Absolute Layout manager will generate null layouts instead of Absolute layouts
Grid X	Size of grid for AbsoluteLayout in the X axis

Property	***Description***
Grid Y	Size of grid for AbsoluteLayout in the Y axis
Indent AWT Hierarchy	If true, the code generated in `initComponents` () is indented for child components of a container.
Property Editor Search Path	List of packages that are searched for property editors that are to be used in the Form Editor
Property Editors	Explicitly registered editors for certain property types
Show Grid	If true, a grid is displayed in the Form Editor when using Absolute-Layout.
Variables Modifier	The access modifier of variables generated for components on the form (private, package private, protected, or public)
Apply Grid to Position	If true, the position of components is snapped to grid (if a grid is used).
Apply Grid to Size	If true, the size of components is snapped to grid (if a grid is used).
Connection Border Color	Color of components' selection border during connection mode
Drag Border Color	Color of components' drag border during dragging
Selection Border Size	Size (in pixels) of the boxes around a component which mark it as “selected”
Selection Border Color	Color of the boxes around a component which mark it as “selected”.

HTTP Server settings reference

The last two properties in the following table can be found under the **Expert** tab of the property sheet.

Table 35: HTTP Server properties

Property	*Description*
Grant access to	Specifies machines which allows access to the HTTP server. Machines are entered as a comma-separated list of IP addresses.
Host	Host has two possible settings: `Any Host` or `Selected Hosts`. `SelectedHosts` restricts access so that only the machine on which Forte for Java is running and machines specified in that field are allowed access. (default=`Selected Hosts`)
Port	The port number on which the HTTP server operates. (default=8081)
Running	If `True`, the HTTP server is running
Base class path URL	Gives access to internal IDE resources.
Base Repository URL	Gives access to the contents of the Repository.

Java Elements settings reference

Table 36: Java element properties

Property	*Description*
Classes	Display name of classes (using a combination of plain text and substitution codes)
Constructors	Display name of constructors (using a combination of plain text and substitution codes)
Fields	Display name of fields (using a combination of plain text and substitution codes)
Initializers	Display name of initializers (using a combination of plain text and substitution codes)
Interfaces	Display name of interfaces (using a combination of plain text and substitution codes)
Methods	Display name of methods (using a combination of plain text and substitution codes)

Table 37: Substitution Codes for the Java Elements properties

Substitution Code	Type of substituted text
{m}	Element's modifiers (for all elements except initializers)
{n}	Element's name (for all elements except initializers)
{C}	Name of class with all outerclasses (for classes and interfaces only)
{f}	Full name of element with package (for all elements except initializers)
{t}	Type (for fields only)
{r}	Return type (for methods only)
{s}	Superclass (for classes only)
{c}	Static (for initializers only)
{p}	Parameters with types but not variable names (for constructors and methods)
{a}	Parameters with types *and* names (for constructors and methods only)
{i}	Interfaces (for classes and interfaces only)
{e}	Exceptions (for constructors and methods only)
<initializer>	Initializer

Java Sources settings reference

Table 38: Java Sources properties

Property	*Description*
Automatic parsing delay	Parser auto-start time-out in milliseconds.
Parse class files	If True, class files are parsed for their Source property.
Strings table	Table of substitution keys for templates.

Source synchronization

Table 39: Source synchronization properties

Property	*Description*
Return generation mode	Generate either nothing, an exception, or the string "return null" when creating a new method declared to return a value.
Synchronization enabled	If `False`, all synchronization is turned off.

Object Browser settings reference

Table 40: Object Browser properties

Property	*Description*
Package Filter	Custom property editor available to add, delete, and rename package filters to the Object Browser

Open File Server settings reference

Table 41: Open File Server properties

Property	*Description*
Access Restriction	If set to `Any Host`, people on other machines can open files in your IDE. Set to `Local Host Only` by default.
Port	The port the server runs on - by default 7318.
Running	If True, the server is on.
Quiet Mode	If True, errors from Java about sockets will be suppressed. Useful on Linux with native threads.

Output Window settings reference

Table 42: Output Window properties

Property	*Description*
Background	Background color of the Output Window
Cursor Background	Background color of highlighted text
Cursor Foreground	Color of highlighted text
Font Size	Size of the characters in the Output Window
Foreground	Default text color
Jump Cursor Foreground	Text color for lines of text in the Output Window that are linked to lines in the Editor
Jump Cursor Background	Background color for lines of text in the Output Window that are linked to lines in the Editor
Tab Size	The number of spaces represented by a TAB stroke

Print Settings reference

Table 43: Print Settings properties – general

Property	*Description*
Line Ascent Correction	A multiplier to adjust the spacing between lines
Page Footer Alignment	Options: LEFT, CENTER, and RIGHT
Page Footer Font	Has a custom property editor available allowing you to set font face, style, and size.
Page Footer Format	You may set the footer with a combination of text and the following tags – {0} for page number, {1}for date, and {2} for file name.
Page Header Alignment	Options: LEFT, CENTER, and RIGHT
Page Header Font	Has a custom property editor available allowing you to set font face, style, and size.
Page Header Format	You may set the header with a combination of text and the following tags – {0} for page #, {1}for date, and {2} for file name.
Wrap Lines	If True, lines are wrapped.

Table 44: Print Settings properties – by individual Editor type

Property	*Description*
Print Fonts and Colors	Provides access to a custom property editor where background and foreground colors as well as fonts can be set for various syntax tokens.
Print line numbers	If True, line numbers appear in printouts.

Property Sheet settings reference

Table 45: Property Sheet properties

Property	***Description***
Disabled Property Color	The text color of read-only properties
Display Editable Only	Whether to show only properties that are writable or that have a custom property editor
Painting Style	Indicates if properties are shown always as text, preferably as text, or preferably as graphics.
Plastic	Whether to enable animation of buttons in the property sheet to make the active property more visible
Sorting Mode	Sets the criteria for sorting properties on the Property Sheet – by name, by type, or unsorted.
Value Color	The text color of property values in the Property Sheet

System Settings reference

Table 46: System Settings

Property	***Description***
Confirm Delete	If `True`, a dialog appears to confirm any deletions you make.
Home Page	Home page for the internal Web browser
Look&Feel	Metal, CDEMotif, and Windows options
Proxy Host	Host of the proxy server
Proxy Port	Port number of the proxy server
Show Tips on Startup	True/False
Use Proxy	If True, the proxy server is used

Appendix E

Actions

This table lists and briefly describes each of the actions available in the "actions pool" under `Global Settings / Actions`.

Table 47: Build Actions

Action	*Description*
Build	Force compilation of all objects in selected folder, whether current or not
BuildAll	Build selected folder and all sub-folders, recursively
BuildProject	Force compilation of all objects in selected folder, whether current or not (multiple projects are available using the Projects module)
Compile	Compile the selected object
CompileAll	Compile the selected folder and all sub-folders recursively
Execute	Execute the selected object
NextError	Jump to the next error in the Output Window
PreviousError	Jump to the previous error in the Output Window
SetArguments	Set command line arguments to pass to an application

Action	Description
StopCompile	Halt the current compilation process

Table 48: Debugger Actions

Action	Description
Add BreakpointAction	Add a breakpoint
AddWatch	Add a watch
ConnectAction	Connect Debugger to a process in an already running virtual machine
DebuggerViewAction	Make the Debugger Window the active window
Finish Debugger	Terminate debugging session
Go	Initiate a debugging session
Go To Cursor	Go to current line in the Editor
ResumeDebuggerAction	Resume debugging of the selected threads
StepOut	Halt execution after the current method finishes and control passes to the caller
SuspendDebuggerAction	Suspends the selected threads in the Debugger.
Toggle Breakpoint	Toggle selected breakpoint on or off
Trace Into	Trace into the method the debugger has halted at
Trace Over	Trace over the method the debugger has halted at

Table 49: Edit Actions

Action	Description
Copy	Copy selected object to the clipboard
Cut	Cut the selected object, keeping a copy in the clipboard
Delete	Delete the selected object
Find	Find specified text in Editor
Goto	Go to specified line number in the Editor

Action	Description
Paste	Paste from the clipboard
Redo	Redo undone action
Replace	Replace in text
Undo	Undo last action

Table 50: Form Actions

Action	Description
ComponentInspecto-rAction	Go to the selected component in the Component Inspector
CustomizeLayoutAc-tion	Invoke the customizer dialog for GridBag Layout
DesignModeAction	Put the Form Editor in design mode
EventsAction	Lists events for selected component
GotoEditorAction	Go to the line in the Editor corresponding to the selected component
GotoFormAction	Go to the selected component in the Form Editor window
InstallBeanAction	Install a JavaBean into the Component Palette
PaletteAction	Install the Component Palette
SelectLayoutAction	Set the layout
ShowGridAction	Display a grid in the Form Editor window (for Absolute layout)
TestModeAction	Put the Form Editor in test mode

Table 51: Help Actions

Action	Description
About	Display the About dialog box
BookmarksAction	Show the bookmarks submenu
Help	Browse the documentation in the web browser
Tip Of The Day	Display the Tip of the Day dialog

Table 52: System Actions

Action	*Description*
AddDirectory	Mount a new file system under the Repository
AddJarArchive	Mount a Jar archive under the Repository
CustomizeBean	Customize properties of a JavaBean
Exit	Exit the IDE
FileSystemAction	Invoke the file system submenu for the given module
GarbageCollect	Garbage Collect. By default, this item is not installed in any menus or toolbars
Instantiate	Instantiate a class
Move Down	Moves current item down among the parent's children
MoveUp	Moves current item up among the parent's children
New	Create a new object
NewAction	Create a new object using an existing template
Open	Open an object
OpenExplorer	Open a new instance of the Explorer
Print	Print the file active in the Editor or selected in the Explorer
Refresh	Refreshes state of a component
RemoveFromReposi-tory	Unmount an mounted file system
Rename	Rename selected object
Reorder	Change order of subnodes (subcomponents) of selected item (container)
Save	Save current object
SaveAll	Save all open objects
SaveAsTemplate	Save a copy of the object as a template in the Templates hierarchy
SetDefaultValue	Set the default value for the property.
Tools	Show the Tools menu (menu-only action)
View	View an object (for example, by launching the HTML browser)

Table 53: View Actions

Action	Description
CloseView	Close the active window
Customize	Invoke a JavaBean's Customizer
EditorWindow	Open a new instance of the Editor Window
Execution Window	Open a new instance of the Execution Window
Explore From Here	Open a new instance of the Explorer, with the selected node as the root
ExplorerWindow	Open a new instance of the Explorer
GlobalPropertiesAc-tion	Open a Property Sheet window to display the property sheet for any sub-sequently selected object
LookAndFeel	List and switch to Look & Feels
NextWorkspace	Switch to the next workspace
OutputWindow	Open a new instance of the Output Window
PreviousWorkspace	Switch to the previous workspace
PropertiesAction	Open the property sheet in a separate window for the selected object
StatusLine	Installs the IDE’s status line (available only for toolbars)
WebBrowserWindow	Open a new instance of the Web Browser window
Workspaces	List and switch Workspaces

Table 54: Window Actions

Action	Description
CloneView	Clone the current view
DockInto	Dock the current window into a separate window or into a multi-tab win-dow
NextTab	Switch to the next tab in a multi-tab window
OpenedWindows	Lists opened windows in the Windows menu (available only for menus)
PreviousTab	Switch to the previous tab in a multi-tab window
UndockWindow	Undock the current tab from the MultiWindow

:

:

: