

**IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume IV**

Version 3.5

Document Number S33H-5042-00

IBM VisualAge for C++ for Windows

S33H-5042-00

Open Class Library Reference
Volume IV

Version 3.5

IBM

IBM VisualAge for C++ for Windows

S33H-5042-00

**Open Class Library Reference
Volume IV**

Version 3.5

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page x.

First Edition (March 1996)

This edition applies to Version 3.5 of IBM VisualAge for C++ and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order books through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada. M3C 1H7

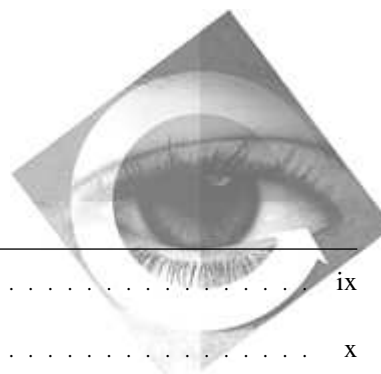
You can also send your comments by facsimile (attention: RCF Coordinator), or, you can send your comments electronically to IBM. See “Communicating Your Comments to IBM” for a description of the methods. This page immediately precedes the Readers’ Comment Form at the back of this publication.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1992, 1996. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents



Part 1. About this Book	ix
Notices	x
Programming Interface Information	x
Trademarks	xi
About This Book	xii
Who Should Use This Book	xii
Conventions Used in This Book	xii
How This Reference is Organized	xiii
About the User Interface Class Library	xiv
User Interface Class Library Conventions	xvii
File Names	xvii
Class Names and Member Names	xix
Function Return Types and Function Arguments	xix
Using Obsolete or Ignored Member Functions	xx
A Note about Samples and Examples	xxii
Part 2. Description of Classes	1
Class Hierarchy by Category	2
IComponent	7
IComponentStationery	13
IComponentStationeryFor	20
IDM	23
IDMCnrItem	35
IDMEFItem	41
IDMEvent	46
IDMHandler	48
IDMImage	60

IDMImage::Style	70
IDMItem	72
IDMItemProvider	102
IDMItemProviderFor	107
IDMMenuItem	110
IDMMLItem	115
IDMOperation	120
IDMRenderer	131
IDMSourceBeginEvent	136
IDMSourceDiscardEvent	139
IDMSourceEndEvent	142
IDMSourceHandler	145
IDMSourceOperation	153
IDMSourcePrepareEvent	160
IDMSourcePrintEvent	164
IDMSourceRenderer	167
IDMSourceRenderEvent	174
IDMTargetDropEvent	179
IDMTargetEndEvent	183
IDMTargetEnterEvent	187
IDMTargetEvent	192
IDMTargetHandler	195

IDMTargetHelpEvent	202
IDMTargetLeaveEvent	205
IDMTargetOperation	207
IDMTargetRenderer	213
IDMTBarButtonItem	220
IDMToolBarItem	227
IDocumentStorage	231
IEmbeddedComponent	235
IEmbedderModel	243
IFlatFileStorage	250
IFont	253
IFont::FaceNameCursor	269
IFont::PointSizeCursor	272
IG3PointArc	275
IGArc	280
IGBitmap	285
IGChord	302
IGEllipse	305
IGLine	309
IGList	313
IGList::Cursor	321
IGPie	325

IGPolygon	330
IGPolyline	334
IGrabHandles	339
IGraphic	343
IGraphicBundle	353
IGraphicContext	379
IGRectangle	406
IGRegion	410
IGString	423
IGUIBundle	429
IMGrabbable	437
IMM24FramesPerSecondTime	439
IMM25FramesPerSecondTime	442
IMM30FramesPerSecondTime	445
IMMAmpMixer	448
IMMAudioBuffer	459
IMMAudioCD	467
IMMAudioCDContents	484
IMMAudioCDContents::Cursor	489
IMMCDXA	493
IMMConfigurableAudio	498
IMMCuePointEvent	509

IMMDevice	512
IMMDeviceEvent	547
IMMDeviceHandler	550
IMMDeviceNotifyHandler	554
IMMDigitalVideo	557
IMMErrorInfo	579
IMMFileMedia	584
IMMHourMinSecFrameTime	592
IMMHourMinSecTime	597
IMMMasterAudio	601
IMMMillisecondTime	606
IMMMinSecFrameTime	609
IMMNotifyEvent	613
IMMPassDeviceEvent	617
IMMPlayableDevice	620
IMMPlayerPanel	631
IMMPlayerPanelHandler	638
IMMPositionChangeEvent	643
IMMRecordable	646
IMMRemovableMedia	657
IMMRemovableMediaHandler	665
IMMRemovableMediaNotifyHandler	669

IMMSequencer	672
IMMSpeed	677
IMMTime	680
IMMTrackMinSecFrameTime	689
IMMWaveAudio	694
IModel	703
IRegionHandle	709
IStructuredStorage	711
ITransformMatrix	715
IView	722
Appendix A. Classes and Header Files Cross-Reference Tables	729
Glossary	764
Bibliography	781
Index	782

Part 1. About this Book

Notices	x
Programming Interface Information	x
Trademarks	xi
 About This Book	 xii
Who Should Use This Book	xii
Conventions Used in This Book	xii
How This Reference is Organized	xiii
About the User Interface Class Library	xiv
User Interface Class Library Conventions	xvii
File Names	xvii
Class Names and Member Names	xix
Function Return Types and Function Arguments	xix
Using Obsolete or Ignored Member Functions	xx
A Note about Samples and Examples	xxii



Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, is the user's responsibility.

IBM might have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This book is intended to help you develop applications using the C++ class libraries provided with VisualAge for C++. This publication documents General-Use Programming Interface and Associated Guidance Information provided by C++

General-Use programming interfaces allow the customer to write programs that obtain the services of VisualAge for C++.

Trademarks

The following terms are trademarks of the International Business Machine Corporation in the United States or other countries or both:

AIX	OS/2 Warp
AIX/6000	Personal System/2
C Set ++	Presentation Manager
CUA	PS/2
IBM	System Object Model
IBMLink	VisualAge
Operating System/2	WorkFrame
OS/2	

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

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

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

IBM's VisualAge products and services are not associated with or sponsored by Visual Edge Software, Ltd..

Who Should Use This Book

About This Book

The *IBM VisualAge for C++ Open Class Library Reference* (hereafter called *Open Class Library Reference*) provides a reference to all the classes for the User Interface Class Library, which is part of the VisualAge for C++ product.

The User Interface Class Library classes are divided between Volume II, Volume III, and Volume IV of the *Open Class Library Reference*. See “How This Reference is Organized” on page xiii for information on the content of the reference volumes. Within each volume, the classes are listed alphabetically for easy retrievability.

Use this reference if you want to build applications with graphical user interfaces.

Who Should Use This Book

This book is intended for application programmers who want to develop portable C++ applications using the User Interface Class Library, part of the IBM Open Class Library product. You should be familiar with the Windows operating system, the AIX operating system, or the IBM Operating System/2* (OS/2*) operating system and OS/2 Presentation Manager* (PM). You should also be familiar with the C++ programming language.

If you are not familiar with the Windows operating system, refer to your Windows documentation. If you are not familiar with OS/2, refer to the programming guides in the IBM OS/2 Technical Library.

For detailed information about C++ language keywords and definitions, refer to the *VisualAge for C++: Language Reference*.

Refer to the *IBM Open Class Library User's Guide* for information on how to use the User Interface Class Library.

If you are not familiar with the IBM AIX Version 4.1 Operating System, refer to *IBM AIX Version 4.1 System User's Guide* before reading this book.

Conventions Used in This Book

New or unfamiliar terms appear in *italics* on their first occurrence in this reference. Acronyms and abbreviations are spelled out the first time they are used. The glossary also contains definitions for new terms, acronyms, and abbreviations.

Static members are shown in bold within the cross-reference tables.

Who Should Use This Book

References to pages within this volume are indicated by the page number placed in parentheses.

How This Reference is Organized

The User Interface Class Library classes are grouped into major categories. Volume II contains the classes grouped in the following categories:

- Application Control Classes
- Base Window, Menu, Handler, and Event Classes
- Standard Control Classes

Volume III contains the classes grouped in the following categories:

- Advanced Control, Dialog, and their Handler Classes
- Dynamic Data Exchange Classes

Volume IV contains the classes grouped in the following categories:

- Two-Dimensional Graphic Classes
- Compound Document Framework
- Direct Manipulation Classes
- Multimedia Classes

Volume IV also contains cross-reference information about the classes in Volume II and Volume III.

Refer to “About the User Interface Class Library” on page xiv for details on grouping categories. For information on class descriptions by name, see Part 2, “Description of Classes.”

In this book, classes, member functions, and enumerations might have the following sections:



Portability Considerations



Motif Information



Presentation Manager Information



Windows Information



Win32s Information

These sections contain information that is specific to a particular platform or suggestions for developing portable applications. Each section is designated by the icon shown so that information for a particular platform is easy to find.

Note: The platform specific notes in the Windows Information section (denoted by the Win icon) relates to all Windows platforms. Any additional information

About the User Interface Class Library

for using Win32s is documented in the Win32s Information section (denoted by the Win32s icon).

Note that some members have a Platform Support Table in the reference. If a member function is overloaded, one overload might have one condition, while a second overload has a different condition. For example, in IBaseSpinButton::initialize, one implementation is for Windows, AIX and OS/2, while another implementation is for Windows and OS/2 only. Also, some functions with only one implementation might be supported on only one platform. This is indicated in the Platform Support Table as well. This table lists each platform in the heading and the table entries explain if the function is supported (Y), not supported (N), or silently ignored (I).

For example:

1	virtual void initialize();	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
2	void initialize(unsigned long windowId, const IWindowHandle& parent, const IWindowHandle& owner, unsigned long style, const IRectangle& initial);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>

Volume III also contains a number of cross-references likely to be of use to you in your development activity.

Appendix A, “Classes and Header Files Cross-Reference” contains two cross-reference tables. You can use the first to find the name of the header file that contains a particular class. The second helps you find what classes are in a single header file.

In “Bibliography” you can find books related to the VisualAge for C++ User Interface Class Library, C and C++ books, as well as other useful publications.

About the User Interface Class Library

The User Interface Class Library is one of the class libraries included in the IBM VisualAge for C++ for Windows product. It is a C++ class library that simplifies how you develop Windows, OS/2, and Motif applications with graphical user interfaces (GUI). The User Interface Class Library provides classes that you can use in your applications and reuse to extend your applications.

You can use the User Interface Class Library classes to build applications that simulate both the Common User Access (CUA) workplace look-and-feel to take

About the User Interface Class Library

advantage of PM features on the OS/2 operating system, and the native controls provided by the Windows operating system. You can create applications for Windows NT, Windows 95, and for the Win32s platforms. To run applications on Win32s systems, we require Win32s 1.30 or later with OLE support. You must use Windows 95 or Windows NT for your development environment to use the IBM VisualAge for C++ for Windows product.

Using the IBM C Set ++ for AIX product, you can use the User Interface Class Library to develop Motif** applications so that you can take advantage of Motif 1.2 features.

You can also use the User Interface Class Library to develop applications that are portable between the OS/2, Motif, and Windows operating systems.

The User Interface Class Library contains over 400 classes and over 4000 member functions. To assist you in learning about the classes and to guide you as you start developing portable applications, we organized the classes into the following basic categories:

- Application Control Classes
- Data Type, Stream, and Exception Classes
- Base Window, Menu, Handler, and Event Classes
- Standard Control Classes
- Advanced Control, Dialog, and Handler Classes
- Direct Manipulation Classes
- Dynamic Data Exchange Classes
- Two-Dimensional Graphic Classes
- Multimedia Classes
- Compound Document Framework Classes

Application Control Classes: Provide support for the application, threads, timers, profiles, and resources used by the applications you develop.

Data Type, Stream, and Exception Classes: Provide support for the exceptions, trace output, messages, strings, notifications, and window geometry used by the applications you develop.

These classes model basic data types, such as strings, points, and rectangles. These classes hide the structure of the data, while providing the capability to access and alter the data. In addition, a set of handle classes are provided to access window or application-specific handles.

About the User Interface Class Library

Base Window, Menu, Handler, and Event Classes: Provide support for the basic windows, handlers, events, and menus used by the applications you develop.

These classes encapsulate the basic graphical building blocks that are used to construct application windows. This encapsulation allows window position and appearance (parent windows) to be separated from event-handling (owner windows).

These classes encapsulate the user's interaction with application windows. The library creates event objects as a result of some action by the user or by other applications. These event objects contain information about what occurred; they are passed to handler objects for processing. Each window has some default event-processing; however, the application can create instances of the handler classes to process certain event objects to override the default behavior.

Use handlers to override or augment a control's default behavior. For example, use a handler when you want to take some action based on user input, such as selecting a button or list box item.

Standard Control Classes: Provide support for the standard controls such as entry field, static text, and buttons used by the applications you develop.

Advanced Control, Dialog, and Handler Classes: Provide support for the advanced controls such as container, notebook, tool bar, and the font and file dialogs used by the applications you develop.

Direct Manipulation Classes: Provide support for the direct manipulation used by the applications you develop.

Dynamic Data Exchange Classes: Provide support for the Dynamic Data Exchange (DDE) used by the applications you develop.

Two-Dimensional Graphic Classes: Provide support for the 2D graphic elements used by the applications you develop.

Multimedia Classes: Provide support for the multimedia devices and controls used by the applications you develop.

Compound Document Framework Classes: The Compound Document Framework is comprised of a set of classes and pre-defined collaborations among the classes which provide you with a large portion of the functionality required to create OLE-enabled applications. Use the framework to create both OLE-enabled container and server applications.

User Interface Class Library Conventions

This section describes the User Interface Class Library conventions for the following:

- File names
- Class names and member names
- Function return types
- Function arguments
- Examples
- Obsolete and ignored functions

File Names

File names in OS/2 and Windows have a maximum of eight characters. All files provided by the User Interface Class Library begin with the letter “I” for IBM, for example, IAPP.HPP. All file names in Motif are case sensitive and the User Interface Class Library uses lowercase letters, for example, iapp.hpp.

The following table lists the Windows and OS/2 file names, file extensions, and a brief description.

File Name and Extension	Description
Ixxxxxx.CPP	Source code
Ixxxxxx.H	Constant definitions file
Ixxxxxx.HPP	Class interface file
Ixxxxxx.INL	Inline functions

The following table lists the Motif file names, file extensions, and a brief description.

File Name and Extension	Description
ixxxxx.cpp	Source code
ixxxxx.hpp	Class interface file
ixxxxx.h	Constant definitions file
ixxxxx.inl	Inline functions

Note: When you use the .cpp extension with IBM C Set ++ for AIX, you need to include the -+ compiler option for your file to be treated as C++ code, not C code.

About the User Interface Class Library



Refer to the appendix for cross-reference tables for the header files and the classes they contain.

The IBM Open Class Library product files for this release begin with the letters “CPP.” The following table lists some file names, file extensions, and a brief description.

File Name and Extension	Description
CPPWO*.LIB	Import library files for each DLL.
CPPWO*.DLL	Multithreaded dynamic-link library files containing the Open Class Library classes (User Interface, Collection, Data Types, and Exception classes).
CPPWO*.DEF	Import module-definition files used to rebuild the DLLs. file.
CPPWO*.RSP	Linker and compiler response files to identify the object modules contained in the DLLs.
CPPWOC3.LIB	Static object library.
CPPWOR3U.DLL	Contains the resources used with tool bars and other User Interface Class Library classes.
CPPBRS.NDX	Index for online help (contains class::member references).
CPP*.INF	Online help and online documentation files.
CPPWOC3U.MSG	Exception messages for the User Interface Classes.
DDE4C01E.MSG	Exception messages for the Collection Classes.

For the Windows platform, the Open Class Library DLLs are as follows:

CPPWOB3I.DLL Multithreaded dynamic link library file containing the base library classes (Collection, Data Types, and Exceptions).

CPPWOD3I.DLL Multithreaded dynamic link library file containing the Dynamic Data Exchange classes.

CPPWOF3I.DLL Multithreaded dynamic link library file containing the Document Framework classes.

CPPWOM3I.DLL Multithreaded dynamic link library file containing the Multimedia classes

CPPWOT3.DLL Multithreaded dynamic link library file containing the CUA '91 controls.

About the User Interface Class Library

CPPWOU3L.DLL Multithreaded dynamic link library file containing the User Interface Base Library classes.

The following table lists some other file names used by the User Interface Class Library for AIX, their extensions, and a brief description.

File Name and Extension	Description	Installed Location
libbmui.a	Static object library for User Interface Class Library	/usr/lpp/xlC/lib
libbmuis.a	Shared library for User Interface Class Library	/usr/lpp/xlC/lib
ibmcl.cat	Exception message catalog file	/usr/lib/nls/msg/\$LANG

Note: \$LANG represents language; for example, if the language is US English, substitute En_US for \$LANG.

Class Names and Member Names

The following rules were used for naming the User Interface Class Library classes and members:

- Type names begin with a capital letter.
- Global type names begin with the letter “I,” as in `ICurrentApplication`.
- Member names, including member functions, member data, and enumerations, begin with lowercase letters, as in `autoSize` data member.

Note: In this book, single-word member functions have `ClassName::` added to them; for example, the member function “show” appears as `IWindow::show`.

Function Return Types and Function Arguments

To follow the User Interface Class Library conventions, pass objects by reference preferable as *const* references and return objects by value rather than by reference. Pass objects by pointer rather than by reference when you want a parameter to use its default.

Function return types for the various functions are as follows:

- A Boolean (true or false). Use `IBase::Boolean` because it is portable between Windows, OS/2, and Motif. The following is an example of a testing function:

```
IBase::Boolean isValid() const;
```

Note: The User Interface Class Library returns a 0 if false and a nonzero if true, so do not test for the following:

```
isValid() == true
```

About the User Interface Class Library

Instead, use the following:

```
if(isValid)
```

- An object. Accessor functions typically return an object. An *accessor* returns information about the elements of a data type. The following example returns a pointer to an IWindow object.

```
IWindow* owner();           //Returns a pointer to an object
```

- An object reference. Functions that act on an object return a reference to the object on which they were called. For example:

```
IWindow& hide();
```

This lets you chain function calls together, as shown in the following example:

```
window.moveTo(IPoint(10,10)).show();
```

Function arguments are passed in the following ways:

- Built-in types (integers or doubles, for example) and enumerations are passed in by value.
- Objects are passed by reference. If the argument is not modified by the function, it is passed as a const reference.
- Optional objects are passed by pointer. This allows a 0 pointer to signify that no object is being passed.
- IWindow objects are passed by pointer.
- IContainerObjects are passed by pointer.
- Strings are passed as a const char *. This enables you to pass either an IString object or an array of characters.

Using Obsolete or Ignored Member Functions

The following sections define obsolete and ignored and explain how to use these functions in the User Interface Class Library. To develop portable applications, you should be aware of these conventions.

Obsolete Functions

As the Open Class Library functionality increases, there are situations where we must change the interface to improve the quality and design. We identify the interface that is obsoleted and provide this information so you can migrate to replacement classes and functions.

In `ibase.hpp` we define a set of macros: one to conditionally define the obsolete level for this version of the library, and one for each version of the library in which we

About the User Interface Class Library

have obsoleted. For example, the first version we obsoleted interface in was 310, and the second was 320.

```
#define IC_OBSOLETE_1    310

#define IC_OBSOLETE_2    400

#ifndef IC_OBSOLETE
#ifdef IC_WIN
#define IC_OBSOLETE 320
#endif
#ifdef IC_PM
#define IC_OBSOLETE 310
#endif
#endif
```

Obsolete interface is then wrapped as follows:

```
#if (IC_OBSOLETE <= IC_OBSOLETE_1)
    // obsolete interface here
#endif // IC_OBSOLETE
```

Notice that `IC_OBSOLETE` is conditionally defined in `ibase.hpp` so that you change the define. You can easily identify the obsolete interfaces you are currently using by defining `IC_OBSOLETE` to be greater than any of the obsolete levels. For example, you define `IC_OBSOLETE` to be 500 based on the preceding values, you receive compile errors for each obsolete function used in your code.

There are several important guidelines regarding obsolete functions:

- Usually the implementation of an obsolete function calls the function that has replace it.
- Typically, we remove the interface obsoleted in a version of library in the next major release of the library. We do not document obsoleted interface in the main body of the reference manual. Instead, it is documented in a section which identifies obsolete interface and replacement classes and functions if they are available.

Ignored Functions

When a function cannot be implemented on a particular platform but its usage can be safely ignored, we do so to achieve a higher degree of portability. This is only done when subsequent function calls to the object are not affected by the fact that the function call was ignored.

In order to identify these functions we wrap them in `#ifdefs`. The valid values for the no-op macros are the six `IC_XXXXXX` macros defined with the string `"_FLAGNOP"` appended to them. They are used only with the `#ifndef #endif` preprocessor directive.

Samples and Examples

For example, `IFont::setFontShear()` is a no-op under Motif. Here's what is coded in `IFont.hpp`:

```
#ifndef IC_MOTIF_FLAGNOP
IFont
    &setFontShear( );
#endif
```

These no-op macros are left in the headers so you can identify which functions are ignored. If you need to identify which ignored functions you are using in your code, define the no-op flag for the current platform and compile the code. For example, to find the ignored functions on the Windows platform, compile your code as follows:

```
icc -dIC_WIN_FLAGNOP sampcode.cpp
```

You receive error messages from the compiler for each ignored function used.

Note: Code compiled with one of these macros defined cannot be run because the generated code does not match the shipped DLLs. These macros are only provided to assist you in identifying obsolete functions and code must be recompiled without any of these macros defined to be executable.

A Note about Samples and Examples

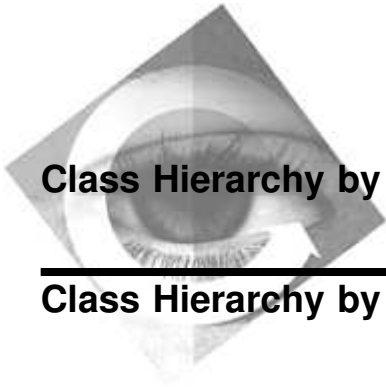
This *Open Class Library Reference* contains examples and references to samples. Samples, including the Hello World sample application, are shipped with the User Interface Class Library product in the following directory:

```
\ibmcpw\samples\ioc
```

Examples, on the other hand, exist only in the *Open Class Library Reference*.

The samples and examples in the User Interface Class Library books explain elements of the User Interface Class Library. They are coded in a simple style. They do not try to conserve storage, check for errors, achieve fast run times, or demonstrate all possible uses of a particular class or function. The samples and examples are instructive but not necessarily intended for productive use.

Part 2. Description of Classes



Class Hierarchy by Category

Class Hierarchy by Category

Direct Manipulation Classes

Provide support for the direct manipulation used by the applications you develop.

```
IDM
IBase
├─ IBitFlag
│   └─ IDMImage::Style
├─ IVBase
│   ├── IDMImage
│   ├── IDMItemProvider
│   │   └─ IDMItemProviderFor
│   ├── IDMRenderer
│   │   ├── IDMSourceRenderer
│   │   └─ IDMTargetRenderer
│   ├── IEvent
│   │   └─ IDMEvent
│   │       ├── IDMSourceBeginEvent
│   │       ├── IDMSourceDiscardEvent
│   │       ├── IDMSourceEndEvent
│   │       ├── IDMSourcePrintEvent
│   │       ├── IDMSourceRenderEvent
│   │       │   └─ IDMSourcePrepareEvent
│   │       ├── IDMTargetEndEvent
│   │       ├── IDMTargetEvent
│   │       │   ├── IDMTargetDropEvent
│   │       │   ├── IDMTargetEnterEvent
│   │       │   └─ IDMTargetLeaveEvent
│   │       └─ IDMTargetHelpEvent
│   ├── IHandler
│   │   └─ IDMHandler
│   │       ├── IDMSourceHandler
│   │       └─ IDMTargetHandler
│   └─ IRefCounted
│       ├── IDMItem
│       │   ├── IDMCnrItem
│       │   ├── IDMEFItem
│       │   ├── IDMMenuItem
│       │   ├── IDMMLEItem
│       │   ├── IDMTBarButtonItem
│       │   └─ IDMToolBarItem
│       └─ IDMOperation
│           ├── IDMSourceOperation
│           └─ IDMTargetOperation
```

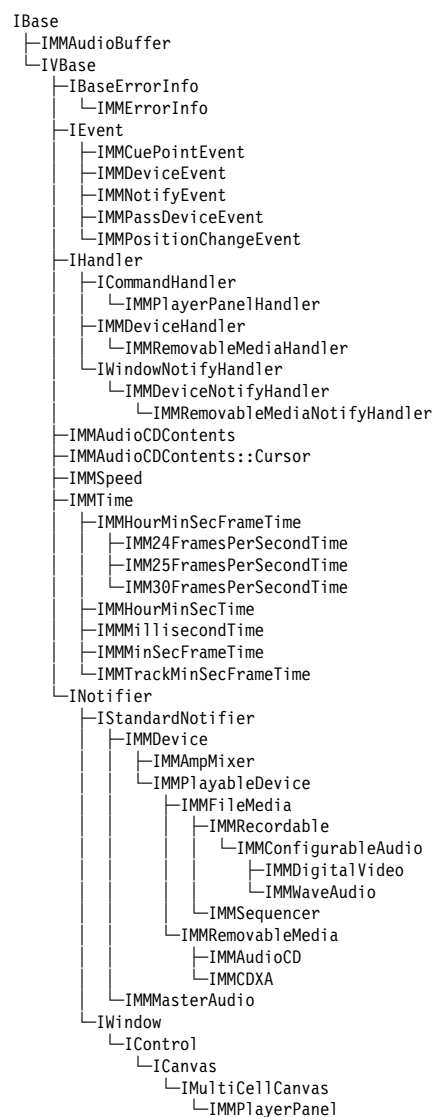
2D Graphic Classes

Provide support for the 2D graphic elements used by the applications you develop.

```
IBase
├── IGraphicBundle
├── IHandle
│   └── IRegionHandle
├── ITransformMatrix
└── IVBase
    ├── IFont
    ├── IFont::FaceNameCursor
    ├── IFont::PointSizeCursor
    ├── IGList::Cursor
    ├── IGraphic
    │   ├── IG3PointArc
    │   ├── IGArc
    │   ├── IGBitmap
    │   ├── IGEllipse
    │   ├── IGLine
    │   ├── IGList
    │   ├── IGPie
    │   │   └── IGChord
    │   ├── IGPolyline
    │   │   └── IGPolygon
    │   ├── IGRectangle
    │   ├── IGRRegion
    │   └── IGString
    └── IGraphicContext
```

Multimedia Classes

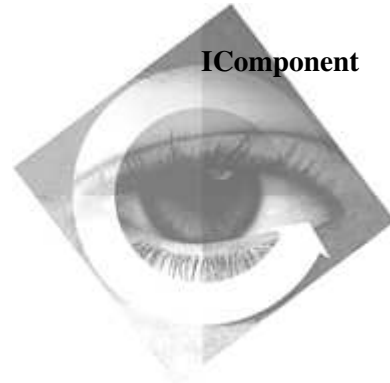
Provide support for the multimedia devices and controls used by the applications you develop.



Component Framework Classes

Provide support for the component document framework classes.

```
IBase
├─ IBase
│   ├─ IComponent
│   ├─ IComponentStationery
│   │   └─ IComponentStationeryFor
│   ├─ IDocumentStorage
│   │   ├─ IFlatFileStorage
│   │   └─ IStructuredStorage
│   ├─ IEmbeddedComponent
│   ├─ IGrabHandles
│   ├─ IModel
│   │   └─ IEmbedderModel
│   └─ INotifier
│       ├─ IWindow
│       │   └─ IControl
│       │       └─ ICanvas
│       │           └─ IView
IGUIBundle
IMGrabbable
```

IconComponent

Derivation IBase
IVBase
IconComponent

Inherited By None.

Header File iconcompn.h

Members	Member	Page	Member	Page
	Constructor	8	isEmbedded	9
	adoptModel	8	isStructuredStorage	10
	bundle	9	model	9
	documentName	9	notifier	9
	handleGetArea	11	orphanModel	8
	handleSetArea	11	setDocumentName	11
	initialize	10	storage	10
	isContainer	9	~IconComponent	8

IconComponent objects act as a wrapper for a document model, and provide the necessary interfaces for communicating with other applications through OLE. IconComponent objects, or components, are created by the framework component stationery. Components are used to manage:

- The model
- Storage for the model

The component adopts the model upon creation, and is responsible for its destruction.

The framework's component stationery creates components as follows:

1. The component stationery receives a request for an OLE server object.
2. IconComponentStationery::createComponent (p. 15) creates an IconComponent object (unless you override the createComponent function in a subclass of IconComponentStationery).
3. A call to IconComponentStationery::createModel (p. 15) creates the model. If the IconComponentStationeryFor (p. 20) template class is used, the created model is of the type specified in the stationery's template parameter.
4. A call to IconComponent::initialize (bundle, model) associates the model with a GUI bundle, and calls IconComponent::adoptModel to adopt the model.

IComponent

The component object exists as long as the application runs, and is destroyed when the application is closed, and the stationery object is destroyed. For more information about component stationeries see IComponentStationery (p. 13).

Public Functions

Adopt/orphan

The framework uses these members to adopt or detach a model.

adoptModel Adopts a new model, and deletes any previous model. This function is called by IComponent::initialize, whenever the component stationery generates a new model (usually when the end-user selects **File ► Open** or **File ► New**).

<pre>virtual void adoptModel(IModel*);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

*IModel** A pointer to the model to be adopted.

orphanModel Detaches a model. The program that calls this function becomes responsible for deleting the model.

<pre>virtual IModel* orphanModel();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Constructors

Use the component stationery to construct objects of this class.

IComponent

<pre>IComponent();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Default constructor. Called by the factory function IComponentStationery::createComponent (p. 15) when the framework creates a component. If you derive a subclass from IComponent, the subclass's constructor must explicitly or implicitly call this default constructor.

~IComponent

<pre>virtual ~IComponent();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

IComponent

Informational

Use these members to obtain information about a component.

bundle Returns a reference (that is, a back-pointer) to the GUI bundle associated with the component.

IGUIBundle& bundle() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--------------------------------	-----------------	----------------	-------------------

documentName

Returns the current document name (that is, the file name), which is set using the setDocumentName function. If no name has been set, returns (Untitled).

const IString& documentName() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---	-----------------	----------------	-------------------

isContainer Returns true if the component is a container, that is, if the model derives from IEmbedderModel (p. 243).

Returns false if the component is a server only, that is, if the model derives from IModel (p. 703).

Boolean isContainer() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------------	-----------------	----------------	-------------------

isEmbedded Returns true if the component is embedded by another component. This function currently returns the same as the IComponentStationery::isEmbedded (p. 14) function (since this is a single-use application they can never be different, since there is only one bundle, one component and one model). Future implementations of the framework will be multi-use enabled so it will be important to use this function and not the one in IComponentStationery.

Boolean isEmbedded() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--------------------------------	-----------------	----------------	-------------------

model Returns a pointer to the component's model, or NULL if no model is attached.

IModel* model() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------	-----------------	----------------	-------------------

notifier Returns a reference to the standard notifier for the component's model. The view, or any other observers, can use the notifier to obtain notification of changes to the model. The notifier is part of the component, rather than part of the model, because

IComponent

the component exists as long as the application is open, whereas the model can be destroyed or replaced at any time (by, for example, the end-user selecting **File ► New** or **File ► Open**).

INotifier& notifier();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------	-----------------	----------------	-------------------

storage Returns a reference to the storage for the component's current model.

IDocumentStorage& storage() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

Initialization

The framework uses this member to initialize a component.

initialize Sets up the component. The component adopts the specified model, and stores a backpointer to the specified bundle. See IDocumentStorage (p. 231) for more information on storage.

virtual void initialize(IGUIBundle& bundle, IModel* adoptedModel);	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--	-----------------	----------------	-------------------

The parameters for this function are as follows:

IGUIBundle&

A reference to the bundle to be associated with the component.

*IModel**

A pointer to the model to be adopted.

isStructuredStorage

Called by the framework to determine if structured storage (or flat file storage) is to be used for storing the data from this component. Returns true if structured storage is to be used; returns false for flat files (the default).

This function applies to servers only, because containers can only use structured storage.

See IDocumentStorage (p. 231) and its subclasses, IFlatFileStorage (p. 250) and IStructuredStorage (p. 711), for more information about the storage classes.

virtual Boolean isStructuredStorage() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---	-----------------	----------------	-------------------

IComponent

Sizing Notification Handlers

The framework calls these members, in response to OLE extent get/set calls, to get and set the size of a component. You can derive a subclass and override these members if their default implementations are unsuitable.

handleGetArea

Returns the current size (in pixels) of the supplied aspect as follows:

kContent	Returns the size of the out-of-place view
kThumbnail	Returns 120 x 120
kIcon	Returns 32 x 32
kDocPrint	Not implemented - returns an arbitrary value

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleGetArea(const EAspect) const;	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

EAspect The aspect to be sized.

handleSetArea

Called by the framework when the size of the component for the specified aspect should be set. The default implementation does nothing.

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleSetArea(const EAspect, const ISize&);	<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

EAspect The aspect to be resized.

ISize& The new size in pixels.

Title Components

Use this member to set the title of a component.

setDocumentName

Sets the document's name to the specified name. If a null string ("") is specified, the name is set to (Untitled). The new name appears in the title bar, and any open servers are notified of the change, so they can update the name in their title bars.

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDocumentName(const IString& name);	<i>Y</i>	<i>N</i>	<i>N</i>

IComponent

The parameter for this function is as follows:

IString& The name to be set for the document.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

EAspect

```
EAspect {  
    kContent = 1,  
    kThumbnail = 2,  
    kIcon = 4,  
    kDocPrint = 8  
};
```



IComponentStationery

Derivation IBase
 IVBase
 IComponentStationery

Inherited By IComponentStationeryFor

Header File istatnry.hpp

Members	Member	Page	Member	Page
	Constructor	19	fileExt	14
	allowInComingCalls	17	initialize	18
	appName	14	isContainer	14
	bundle	16	isEmbedded	14
	createBundle	15	isServer	14
	createComponent	15	messageBox	17
	createFrameWindow	15	rejectInComingCalls	18
	createModel	15	run	18
	createView	16	shortName	14
	enableBusyDialog	17	stationery	16
	enableNotRespondingDialog	17	~IComponentStationery	19

The component stationery, or simply the stationery, is the object that links all portions of the application together. Each part of the application can use stationery functions to access any other part.

The stationery also registers and makes available certain OLE-specific clipboard formats, and groups them into format sets that are used for data transfer between applications.

IComponentStationery is an abstract base class. Use IComponentStationeryFor (p. 20), or create your own subclass that derives from IComponentStationery or IComponentStationeryFor. You can then use the stationery to control the creation of the two main components of the framework: models and views.

This class provides factory functions which the framework uses to create components, models, views, bundles, and frame windows.

You must statically construct a single instance of this class. Construct the instance before WinMain is called, and before the application starts. Once the instance is created, the instance pointer is globally available through the static function call:

IComponentStationery

IComponentStationery::stationery()

Public Functions

Environment

Use these members to obtain information about the component's environment.

appName Returns the application's long name, as entered in the resource file under the key IC_CDF_APP_LONGNAME. The framework registers this name with OLE so that out-of-place servers can display the name on their title bars.

const IString& appName() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
------------------------------------	-----------------	----------------	-------------------

fileExt Returns the file extension as entered in the resource file under the key IC_CDF_FILE_EXT. The framework shows this as the default file extension in all load and save dialog boxes.

const IString& fileExt() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
------------------------------------	-----------------	----------------	-------------------

isContainer Returns true if this application is a container (that is, its model derives from IEmbedderModel).

Boolean isContainer() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------------	-----------------	----------------	-------------------

isEmbedded Returns true if this application is embedded in a container.

Boolean isEmbedded() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--------------------------------	-----------------	----------------	-------------------

isServer Returns true. All Compound Document Framework applications are servers.

Boolean isServer() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
------------------------------	-----------------	----------------	-------------------

shortName Returns the application short name as entered in the resource file under the key IC_CDF_APP_SHORTNAME. Short names are required for OLE applications.

const IString& shortName() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--------------------------------------	-----------------	----------------	-------------------

Factory Members

These members are used to construct the principal objects of the framework. Override these members when you create an application-specific derived class.

createBundle Called by the framework to create the GUI bundle. The GUI bundle is the object that owns and controls the model and view. By default, this function creates an object of class IGUIBundle. See IGUIBundle (p. 429) for more information.

virtual IGUIBundle*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
createBundle() const;	Y	N	N

createComponent

This function creates a new component. The component is the wrapper around the model that allows it to be accessed from other applications, through OLE. By default, this function creates an object of class IComponent. See IComponent (p. 7) for more information.

virtual IComponent*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
createComponent() const;	Y	N	N

createFrameWindow

Called by the framework to create a new frame window. By default, this function creates an object of class IComponentFrameWindow. If the application is a container, the frame window is also registered with OLE as a drop target. The new frame window receives a default menu bar and toolbar from your resource file, using the following IDs:

Menu	IC_CDF_EMBEDDED_MENU_ID (if the application is embedded in a container)
	IC_CDF_STANDALONE_MENU_ID (if the application is not embedded)
Toolbar	IC_CDF_TOOLBAR_ID

In the current release, if you override this function, your override function must still call the base class function to do the actual creation. This process will be more flexible in future releases.

virtual IComponentFrameWindow*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
createFrameWindow();	Y	N	N

createModel The framework calls this pure virtual function to create a new model. There is no default implementation for this function. You can provide an implementation in the following ways:

IComponentStationery

- The preferred way is to use, or derive from, the template class IComponentStationeryFor (p. 20), and use its implementation of this function.
- Derive from this class, and override this function in your derived class.

<pre>virtual IModel* createModel() const = 0;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

createView The framework calls this pure virtual function to create a new view. There is no default implementation for this function. You can provide an implementation in the following ways:

- The preferred way is to use, or derive from, the template class IComponentStationeryFor (p. 20), and use its implementation of this function.
- Derive from this class, and override this function in your derived class.

<pre>virtual IView* createView() const = 0;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Global Objects

Use these members to obtain references to global objects.

bundle Returns a reference to the GUI bundle object. The GUI bundle manages these other framework objects:

- component
- model
- view
- frame window

You can obtain references to these objects from the GUI bundle. See IGUIBundle (p. 429) for more information.

<pre>IGUIBundle& bundle() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

stationery Returns a reference to the global stationery object. Because this is a static function, it can be called without an instantiation. All other stationery functions are accessed through this function. For example:

```
IComponentStationery::stationery().bundle()
```

returns the GUI bundle for the application. See the class description (p. 13) for more information on the global stationery object.

IComponentStationery

```
static IComponentStationery&  
stationery();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Helper Methods

Use these members to control some application behavior.

allowIncomingCalls

Instructs the application to resume accepting incoming OLE messages. This is required after a call to `rejectIncomingCalls` has stopped acceptance of OLE messages.

```
void  
allowIncomingCalls();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

enableBusyDialog

Enables or disables the OLE **Busy Dialog**. This dialog appears when an end-user accesses the application while it is busy (that is, incoming calls are being rejected).

If the dialog is enabled, it gives the end-user three options: **Retry**, **Switch-to**, or **Cancel**. By default, the Busy Dialog is enabled.

If the dialog is disabled, the rejected call is immediately cancelled. See the `rejectIncomingCalls` function.

```
void  
enableBusyDialog( Boolean );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

enableNotRespondingDialog

Enables or disables the OLE **Not Responding** dialog. This dialog appears when the application receives a message while it is waiting for another application. By default, the Not Responding dialog is enabled.

If the dialog is enabled, it gives the end-user two options: **Retry**, or **Switch-to**.

```
void  
enableNotRespondingDialog( Boolean );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

messageBox Displays a modal message box. All incoming calls are automatically rejected while the message box is displayed. Incoming calls are allowed again once the message box is cleared. For a list of the various message box styles, see `IMessageBox` (Vol. II).

IComponentStationery

```
IMessageBox::Response
messageBox(
    const char* msg,
    const IMessageBox::Style& style =
        IMessageBox::informationIcon
        | IMessageBox::moveable
        | IMessageBox::applicationModal
        | IMessageBox::okButton,
    IWindow* owner = NULL );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

rejectIncomingCalls

Instructs the application to reject any incoming OLE messages. Use this function when the application is in a state in which it cannot handle OLE messages (usually because a modal dialog is displayed). Use allowIncomingCalls to re-enable handling of OLE messages.

```
void
rejectIncomingCalls();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Initialization

Use this member to provide application-specific initialization.

initialize

Called by the framework during the application initialization phase (before the component is constructed) so that you can provide any application-specific initialization.

```
virtual Boolean
initialize();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

run

Must be called from your main() function. Performs all necessary framework initialization (including a callout to IComponentStationery::initialize), then runs the application's message processing loop. When the end-user exits the application, this function also handles all necessary shutdown operations.

```
int
run( int argc,
    char* argv [ ] );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Exceptions	
IAccessError	Failed to initialize the underlying compound document runtime environment. Unrecoverable.

IComponentStationery

Constructors

You cannot construct, copy, or destruct objects of this class directly.

~IComponentStationery

```
virtual  
~IComponentStationery();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You cannot construct, copy, or destruct objects of this class directly.

IComponentStationery

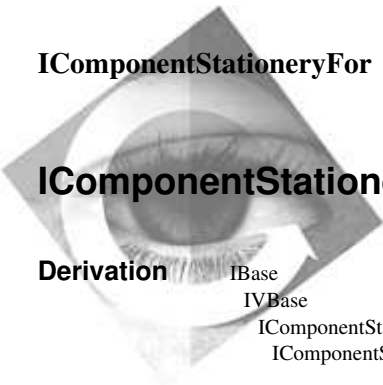
This is a protected constructor. Use IComponentStationeryFor to construct this class.

```
IComponentStationery();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IComponentStationeryFor

IComponentStationeryFor

Derivation IBase
IVBase
IComponentStationery
IComponentStationeryFor

Inherited By None.

Header File istatnry.hpp

Members		Member	Page	Member	Page
		Constructor	20	createView	21
		createModel	21	~IComponentStationeryFor	21

Use this template class to create a component stationery. The component stationery allows the creation of components and other framework objects. To create a stationery, pass a model and a view as template parameters.

Typically, you create the stationery at the bottom of your model .cpp file. For example:

```
IComponentStationeryFor <MyModel, MyView> MyStationery;  
  
creates a stationery instance called MyStationery.
```

For more information about the component stationery, see IComponentStationery (p. 13).

Public Functions

Constructors

You can construct or destruct objects of this class.

IComponentStationeryFor

Constructs the stationery.

IComponentStationeryFor();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	N	N

IComponentStationeryFor

~IComponentStationeryFor

~IComponentStationeryFor();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Factory Functions

The framework uses these members to create new models and views.

createModel Creates a new model. The model is an object of the class you specified when you created the stationery as the first template parameter.

virtual IModel*
createModel() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

createView Creates a new view. The view is an object of the class you specified when you created the stationery as the second template parameter.

virtual IView*
createView() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

IComponentStationery		
allowInComingCalls	createModel	isContainer
appName	createView	isEmbedded
bundle	enableBusyDialog	isServer
createBundle	enableNotRespondingDialog	messageBox
createComponent	fileExt	rejectInComingCalls
createFrameWindow	initialize	run

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IComponentStationeryFor

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDM

IDM

Derivation Inherits from none.

Inherited By None.

Header File idmcomm.hpp

Members				
Member	Page	Member	Page	
any	24	rfSharedMem	26	
binary	24	rfText	26	
binaryData	24	rfUnknown	26	
bitmap	24	rmAny	27	
container	24	rmDiscard	27	
containerObject	24	rmFile	27	
file	24	rmLibrary	27	
icon	25	rmObject	27	
menuItem	25	rmPrint	28	
plainText	25	text	25	
rfAny	26	toolBarButton	25	
rfObject	26	unknown	25	
rfProcess	26			

The IDM structure contains all of the common enumerations and definitions that are shared by the direct manipulation classes.

This structure contains the definitions of the following enumerations:

- Source
- DropIndicator
- RenderCompletion
- DragImageStyle
- DropStyle
- DiscardIndicator
- PrintIndicator

This structure contains the definitions of the following typedefs:

- Type (drag item types)
- RM (rendering mechanisms)
- RF (rendering formats)

This structure defines a static debug support flag for debugging purposes.

IDM

Public Data

Drag Item Types

These strings identify the default drag item types of the User Interface Class Library. Use the drag item type to identify the type of data object that the drag item represents. For example, use `IDM::containerObject` to identify a container object as the data object that the drag item represents.

Note: If the default types do not properly describe the type of data that your drag items represent, you can define new types.

any	Drag item represents any data object. This type combines all of the other types.			
	<code>static Type any;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
binary	Drag item represents a generic binary object.			
	<code>static Type binary;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
binaryData	Drag item represents a binary data object.			
	<code>static Type binaryData;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
bitmap	Drag item represents a bitmap object.			
	<code>static Type bitmap;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
container	Drag item represents a container control object.			
	<code>static Type container;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
containerObject	Drag item represents a container object.			
	<code>static Type containerObject;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
file	Drag item represents a file object.			

IDM

	static Type file;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
icon	Drag item represents an icon object.			
	static Type icon;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
menuItem	Drag item represents a menu item object.			
	static Type menuItem;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
plainText	Drag item represents a plain text object.			
	static Type plainText;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
text	Drag item represents a generic text object.			
	static Type text;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
toolBarButton	Drag item represents a tool bar button object.			
	static Type toolBarButton;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
unknown	Drag item represents an unknown object.			
	static Type unknown;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

Rendering Formats

Use these strings to query or set rendering formats that are supported by the User Interface Class Library.

Note: If the default formats do not properly support your drag items, you can define new formats. See the *User's Guide* for a description of how to implement support for new rendering formats.

IDM

rfAny

Drag item supports any rendering format. This format combines all of the other formats. The User Interface Class Library does not provide default support for this format.

static RF	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rfAny;	<i>Y</i>	<i>Y</i>	<i>N</i>

rfObject

Drag item supports the Workplace Shell's object rendering format. The User Interface Class Library does not provide default support for this format.

static RF	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rfObject;	<i>Y</i>	<i>Y</i>	<i>N</i>

rfProcess

Drag item supports the process rendering format. The process rendering format is created by appending the equal sign and the process ID to this string. See the static member function `IDMItem::rfForThisProcess` (p. 93) for more information.

The User Interface Class Library provides default support for this format.

static RF	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rfProcess;	<i>Y</i>	<i>Y</i>	<i>N</i>

rfSharedMem

Drag item supports the shared memory rendering format. The User Interface Class Library provides default support for this format.

static RF	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rfSharedMem;	<i>Y</i>	<i>Y</i>	<i>N</i>

rfText

Drag item supports the text rendering format. The text rendering format is used if the text length is less than or equal to 255 bytes, if the text does not contain embedded NULL characters, and if the `IDM::rmFile` (p. 27) rendering mechanism is not added in a derived class.

The User Interface Class Library provides default support for this format.

static RF	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rfText;	<i>Y</i>	<i>Y</i>	<i>N</i>

rfUnknown

Drag item supports an unknown rendering format. The User Interface Class Library provides default support for this format.

IDM

static RF
rfUnknown;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Rendering Mechanisms

Use these strings to query or set rendering mechanisms that are supported by the User Interface Class Library.

Note: If the default mechanisms do not properly support your drag items, you can define new mechanisms. See the *User's Guide* for a description of how to implement support for new rendering mechanisms.

rmAny Drag item supports any rendering mechanism. This mechanism combines all of the other mechanisms. The User Interface Class Library does not provide default support for this mechanism.

static RM
rmAny;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

rmDiscard Drag item supports the Workplace Shell's rendering mechanism for the discarding (shredding) of objects. The User Interface Class Library does not provide default support for this mechanism.

static RM
rmDiscard;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

rmFile Drag item supports the Workplace Shell's text rendering mechanism. The User Interface Class Library provides default support for this mechanism.

static RM
rmFile;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

rmLibrary Drag item supports the default rendering mechanism of the User Interface Class Library.

static RM
rmLibrary;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

rmObject Drag item supports the Workplace Shell's object rendering mechanism. The User Interface Class Library does not provide default support for this mechanism.

static RM
rmObject;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDM

rmPrint

Drag item supports the Workplace Shell's rendering mechanism for the printing of objects. The User Interface Class Library does not provide default support for this mechanism.

```
static RM
    rmPrint;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Nested Type Definitions

DiscardIndicator

```
DiscardIndicator {
    sourceDiscards = 0x0001,
    targetDiscards = 0x0002,
    abortDiscard = 0x0003
};
```

Use the DiscardIndicator enumeration as an argument to the member function IDMSourceDiscardEvent::setWhoDiscards (p. 140) to set the discard indicator. Enumerations of the possible discard indicator values include the following:

sourceDiscards

The source of the direct manipulation accepts responsibility for discarding.

targetDiscards

The target of the direct manipulation accepts responsibility for discarding.

abortDiscard

Discarding is aborted.

DragImageStyle

```
DragImageStyle {
    stack3AndFade,
    allStacked,
    systemImages
};
```

Use the DragImageStyle enumeration as an argument to the member function IDMSourceOperation::setImageStyle (p. 155) to set the drag image style. Enumerations of the possible drag image styles to use when displaying the drag image include the following:

stack3AndFade

Shows the drag images attached to the first three drag items added to the source operation. A special icon is then shown that simulates the rest of the images fading out. The direction and location of the drag images in

IDM

relation to each other are determined by the return value from `IDMSourceOperation::stackingPercentage` (p. 155). Use of `IDM::stack3AndFade` is optimal when the user can drag more than three drag items. If no images are specified, system images are displayed.

allStacked

Shows the drag images attached to all of the drag items. If no images are specified, system images are displayed.

systemImages

If the direct manipulation involves a single drag item, the `ISystemPointerHandle::singleFile` icon is displayed. If it involves multiple drag items, the `ISystemPointerHandle::multipleFile` icon is displayed. Any drag images attached to the drag items are ignored. This is the default drag image style.

DropIndicator

```
DropIndicator {  
    notOk = 0x0000,  
    ok = 0x0001,  
    operationNotOk = 0x0002,  
    neverOk = 0x0003  
};
```

Use the `DropIndicator` enumeration as an argument to the member function `IDMTargetEnterEvent::setDropIndicator` (p. 190) to set the drop indicator values for a direct manipulation. Use the `DropIndicator` as an argument to the member function `IDMItem::setDropStatus` (p. 77) to set the drop status of individual drag items. Enumerations of the possible drop indicator values include the following:

ok

The drop is allowed.

notOk

The drop is not allowed at this location within the target.

operationNotOk

The drop is not allowed because the drag operation, for example, a copy, is not supported by the target.

neverOk

The drop is never allowed on the target.

DropStyle

```
DropStyle {  
    notContainer = 0x0000,    dropAtPosition = 0x0001, alignVertical = 0x0002,  
    alignFlow = 0x0004,      flowBeside = 0x0008  
};
```

IDM

Use the `DropStyle` enumeration as an argument to the member function `IDMTargetOperation::setStyle` (p. 209) to set the drop style. Enumerations of the possible drop style values include the following:

notContainer

The target of the direct manipulation is not a container. This is the default drop style for noncontainer targets.

dropAtPosition

Drag items are displayed in the container at the position where they are dropped relative to the origins of the drag images that represent the items. This is the default drop style for non-list view containers and list view containers when the pointing device is directly over a container object.

alignVertical

Drag items are aligned along the vertical axis. Use this style in addition to `IDM::DropStyle::alignFlow`.

alignFlow

Drag items are aligned along the window. If `IDM::DropStyle::alignVertical` is not specified, the drag items are aligned along the horizontal axis. This is the default drop style for list view containers when the pointing device is between container objects.

flowBeside

Drag items are aligned beside the window. Use this style in addition to `IDM::DropStyle::alignFlow` or `IDM::DropStyle::alignFlow` and `IDM::DropStyle::alignVertical`.

PrintIndicator

```
PrintIndicator {  
    sourcePrints = 0x0001,  
    targetPrints = 0x0002,  
    abortPrint = 0x0003  
};
```

Use the `PrintIndicator` enumeration as an argument to the member function `IDMSourcePrintEvent::setWhoPrints` (p. 165) to set the print indicator. Enumerations of the possible print indicator values include the following:

sourcePrints

The source of the direct manipulation accepts responsibility for printing.

targetPrints

The target of the direct manipulation accepts responsibility for printing.

abortPrint

Printing is aborted.

RenderCompletion

```
RenderCompletion {
    targetSuccessful = 0x0001, targetFailed = 0x0002,
    retry = 0x0008,          renderOk = 0x0010,
    fail = 0x0020
};
```

Use the `RenderCompletion` enumeration as an argument to the member function `IDMSourceRenderEvent::setCompletion` (p. 176) to set the render completion values for the source. Use the member function `IDMTargetRenderer::informSourceOfCompletion` (p. 215) to set the render completion values for the target. Enumerations of the possible render completion values include the following:

targetSuccessful

The target successfully completed the rendering operation.

targetFailed

The target failed to complete the rendering operation.

retry

The source completed the rendering operation, and the source allows the target to retry its part of the operation if the target fails.

renderOk

The source completed the rendering operation.

fail

The source failed to complete the rendering operation.

Source

```
Source {
    pointingDevice,
    keyboard
};
```

Use the `Source` enumeration as an argument to the member function `IDMOperation::setSource` (p. 125) to set the type of the device that is the source of the direct manipulation. Enumerations of the possible source values include the following:

pointingDevice

The direct manipulation started via a pointing device, such as the mouse.

keyboard

The direct manipulation started via the keyboard.

Type

```
typedef const char * Type;
```

IDM

Definition of the default drag item types of the User Interface Class Library include the following:

any	Drag item represents any data object. This type combines all of the other types.
binary	Drag item represents a generic binary object.
binaryData	Drag item represents a binary data object.
bitmap	Drag item represents a bitmap object.
container	Drag item represents a container control object.
containerObject	Drag item represents a container object.
file	Drag item represents a file object.
icon	Drag item represents an icon object.
menuItem	Drag item represents a menu item object.
plainText	Drag item represents a plain text object.
text	Drag item represents a generic text object.
toolBarButton	Drag item represents a tool bar button object.
unknown	Drag item represents an unknown object.

RF

```
typedef const char * RF;
```

Definition of the default rendering formats of the User Interface Class Library include the following:

rfUnknown	Drag item supports an unknown rendering format. The User Interface Class Library provides default support for this format.
------------------	--

IDM

rfText

Drag item supports the text rendering format. The text rendering format is used if the text length is less than or equal to 255 bytes, if the text does not contain embedded NULL characters, and if the IDM::rmFile (p. 27) rendering mechanism is not added in a derived class. The User Interface Class Library provides default support for this format.

rfProcess

Drag item supports the process rendering format. The process rendering format is created by appending the equal sign and the process ID to this string. See the static member function IDMIItem::rfForThisProcess (p. 93) for more information. The User Interface Class Library provides default support for this format.

rfSharedMem

Drag item supports the shared memory rendering format. The User Interface Class Library provides default support for this format.

rfObject

Drag item supports the Workplace Shell's object rendering format. The User Interface Class Library does not provide default support for this format.

rfAny

Drag item supports any rendering format. This format combines all of the other formats. The User Interface Class Library does not provide default support for this format.

RM

```
typedef const char * RM;
```

Definition of the default rendering mechanisms of the User Interface Class Library include the following:

rmFile

Drag item supports the Workplace Shell's text rendering mechanism. The User Interface Class Library provides default support for this mechanism.

rmObject

Drag item supports the Workplace Shell's object rendering mechanism. The User Interface Class Library does not provide default support for this mechanism.

rmDiscard

Drag item supports the Workplace Shell's rendering mechanism for the discarding (shredding) of objects. The User Interface Class Library does not provide default support for this mechanism.

IDM

rmPrint

Drag item supports the Workplace Shell's rendering mechanism for the printing of objects. The User Interface Class Library does not provide default support for this mechanism.

rmLibrary

Drag item supports the default rendering mechanism of the User Interface Class Library.

rmAny

Drag item supports any rendering mechanism. This mechanism combines all of the other mechanisms. The User Interface Class Library does not provide default support for this mechanism.

IDMCnrItem



Derivation IBase
 IVBase
 IRefCounted
 IDMItem
 IDMCnrItem

Inherited By None.

Header File idmcnrit.hpp

Members	Member	Page	Member	Page
	Constructor	36	sourceContainer	38
	containerId	37	targetDrop	36
	generateSourceItems	37	~IDMCnrItem	36

The IDMCnrItem class represents container-specific items in a direct manipulation. Containers create objects of the IDMCnrItem class when either of the following occurs:

- A direct manipulation begins in a source container
- A container item is dragged over a potential target container during a direct manipulation

In addition to attributes inherited from IDMItem, objects of this class have a pointer to the source container that is the source of the direct manipulation, the source container's identifier (ID), and a pointer to the container object that the container item represents.

This class provides virtual functions that implement direct manipulation support for the container. Derive item classes to support the direct manipulation of container items when the default User Interface Class Library support does not meet your requirements.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMCnrItem

IDMCnrItem

1	IDMCnrItem(IContainerControl* container, IContainerObject* object, IDMSourceOperation* sourceOperation, const ISize& imageOffset);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y N
----------	---	---

container A pointer to the source container control.

object A pointer to the source container object.

sourceOperation

A pointer to the drag source operation object to which this container item object is to be added.

imageOffset

A reference to the offset of the drag image from the pointing device.

Use this constructor to construct container items for the source of a direct manipulation. It is called by the member function, IDMCnrItem::generateSourceItems (p. 37).

2	IDMCnrItem(const IDMItem::Handle& dragItem);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y N
----------	--	---

dragItem A reference to a handle of a generic drag item that is created by the User Interface Class Library.

Use this constructor to construct container items for the target of a direct manipulation. It is called by the member function, IDMItemProviderFor::provideTargetItemFor (p. 108), of the container's item provider for the container item class.

~IDMCnrItem

virtual ~IDMCnrItem();	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y N
---------------------------	---

Drop Processing

Use these members during the drop operation of a direct manipulation.

targetDrop Implements container-item-specific rendering when the container item that represents a container object is dropped on a target container. This function adds the object into the target container. Derived classes should override this function to insert the

IDMCnrItem

associated container object into the target container when the default support does not meet your requirements.

Note: If multiple container items are dropped, this function is called once for each item. The items are processed in the reverse order in which they were added to the source operation in the member function, IDMCnrItem::generateSourceItems (p. 37).

```
virtual Boolean                                     Win PM Motif
    targetDrop( IDMTargetDropEvent& event );         Y  Y  N
```

event A reference to the target drop event.

Exceptions	
InvalidRequest	The drop cannot be completed. A source object must exist for this operation.

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

containerId Returns the source container's ID.

```
virtual unsigned long                               Win PM Motif
    containerId() const;                             Y  Y  N
```

generateSourceItems

Generates container drag items that represent container objects selected in the source container. This function is called by the container's implementation of IDMItemProvider::provideSourceItems (p. 105) in the template class IDMItemProviderFor (p. 107).

The default implementation of this function in this class creates IDMCnrItem objects and adds them to the source operation. If you use the IDMItemProviderFor template class and you have defined a derived item class, you must implement IDMCnrItem::generateSourceItems in your derived container item class. In your implementation, create one or more objects of your derived container item class and call IDMOperation::addItem (p. 123) to add the objects to the source operation.

Note: Use the *sourceOperation* parameter to access IDMOperation::addItem.

```
static Boolean                                     Win PM Motif
    generateSourceItems( IDMSourceOperation* sourceOperation ); Y  Y  N
```

IDMCnrItem

sourceOperation

A pointer to the source operation object.

sourceContainer

Returns a pointer to the source container.

```
virtual IContainerControl*  
    sourceContainer() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDMItem		
addRMF	nativeRF	setRequiresPreparation
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName

IDMCnrItem

IDMItem		
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IDMCnrItem

IBase		
recoverable	unrecoverable	



IDMEFItem

Derivation IBase
 IVBase
 IRefCounted
 IDMItem
 IDMEFItem

Inherited By None.

Header File idmefit.hpp

Members	Member	Page	Member	Page
	Constructor	42	sourceEnd	42
	generateSourceItems	43	targetDrop	43
	object	43	~IDMEFItem	42

The IDMEFItem class represents entry-field-specific items in a direct manipulation. Entry fields create objects of the IDMEFItem class when either of the following occurs:

- A direct manipulation begins in a source entry field
- An entry field item is dragged over a potential target entry field during a direct manipulation

In addition to attributes inherited from IDMItem, objects of this class have a pointer to the text object that the entry field item represents.

This class provides virtual functions that implement direct manipulation support for the entry field. Derive item classes to support the direct manipulation of entry field items when the default User Interface Class Library support does not meet your requirements.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMEFItem

IDMEFItem

1 IDMEFItem(IDMSourceOperation* sourceOperation); Win PM Motif
Y Y N

sourceOperation

A pointer to the drag source operation object to which this entry field item object is to be added.

Use this constructor to construct entry field items for the source of a direct manipulation. It is called by the member function, IDMEFItem::generateSourceItems (p. 43).

Note: This constructor sets the IDM::rmLibrary rendering mechanism and the IDM::rfProcess rendering format. If the text size is less than or equal to 255 bytes and the IDM::rmFile rendering mechanism has not been added in a derived class, the IDM::rfText rendering format is added. Otherwise, the IDM::rfSharedMem rendering format is added.

2 IDMEFItem(const IDMItem::Handle& dragItem); Win PM Motif
Y Y N

dragItem A reference to a handle of a generic drag item that is created by the User Interface Class Library.

Use this constructor to construct entry field items for the target of a direct manipulation. It is called by the member function, IDMIItemProviderFor::provideTargetItemFor (p. 108), of the entry field's item provider for the entry field item class.

~IDMEFItem

virtual Win PM Motif
~IDMEFItem(); Y Y N

Drop Processing

Use these members during the drop operation of a direct manipulation.

sourceEnd Processes a source end event for the entry field item. This function removes the selected text from the source entry field during a move operation.

virtual Boolean Win PM Motif
sourceEnd(IDMSourceEndEvent& event); Y N N

IDMEFItem

event A reference to the source end event.

targetDrop Implements entry-field-item-specific rendering when the entry field item that represents a text object is dropped on a target entry field. This function sets the text into the target entry field. Derived classes should override this function to set the associated text object into the target entry field when the default support does not meet your requirements.

```
virtual Boolean  
    targetDrop( IDMTargetDropEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target drop event.

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

generateSourceItems

Generates entry field drag items that represent text objects. The text object can be the entire contents of the source entry field or a selected portion.

This function is called by the entry field's implementation of `IDMItemProvider::provideSourceItems` (p. 105) in the template class `IDMItemProviderFor` (p. 107).

The default implementation of this function in this class creates `IDMEFItem` objects and adds them to the source operation. If you use the `IDMItemProviderFor` template class and you have defined a derived item class, you must implement `IDMEFItem::generateSourceItems` in your derived entry field item class. In your implementation, create an object of your derived entry field item class and call `IDMOperation::addItem` (p. 123) to add the objects to the source operation.

Note: Use the *sourceOperation* parameter to access `IDMOperation::addItem`.

```
static Boolean  
    generateSourceItems( IDMSourceOperation* sourceOperation );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sourceOperation
 A pointer to the source operation object.

object Returns a pointer to the contents of the `IString` that contains either all of or the selected portion of the source entry field's text. The pointer can be cast to the type `char *` to directly access the text.

IDMEFItem

```
virtual void*
  object() const;
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDMItem		
addRMF	nativeRF	setRequiresPreparation
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IDMEFItem

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IBase		
recoverable	unrecoverable	



Inherited By

IDMSourceBeginEvent	IDMSourceRenderEvent
IDMSourceDiscardEvent	IDMTargetEndEvent
IDMSourceEndEvent	IDMTargetEvent
IDMSourcePrintEvent	IDMTargetHelpEvent

Header File idmevent.hpp

Members	Member	Page
	Constructor	46
	~IDMEvent	47

The IDMEvent class is the base class for all direct manipulation event classes.

Note: Neither the User Interface Class Library nor Presentation Manager provides default processing for drag events, beyond the User Interface Class Library-provided implementation of virtual functions. If you provide your own implementation of these virtual functions, do not assume returning a value of false will cause correct processing to be done for you.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMEvent Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>
<i>event</i>	A reference to the event object.		

IDMEvent

~IDMEvent

virtual

~IDMEvent();

Win

PM

Motif

Y

Y

N

Inherited Public Functions

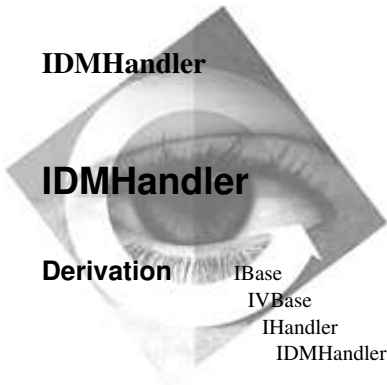
IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By IDMSourceHandler
IDMTargetHandler

Header File idmhndlr.hpp

Members	Member	Page	Member	Page
	Constructor	55	numberOfRenderers	57
	addRenderer	57	removeRenderer	57
	defaultDragHandler	49	renderer	58
	defaultDropTargetHandler	49	replaceRenderer	58
	defaultSourceHandler	49	setDefaultSourceHandler	50
	defaultTargetHandler	49	setDefaultTargetHandler	50
	enableDragDropFor	50	setItemProvider	56
	enableDragFrom	52	setRenderer	58
	enableDropOn	53	~IDMHandler	55
	isContainerControl	56		

The IDMHandler class is the base class for the source handler class, IDMSourceHandler (p. 145), and the target handler class, IDMTargetHandler (p. 195).

This base class provides static functions to enable you to add source and target direct manipulation support to your application's windows and controls.



You are required to use the enabling functions for direct manipulation support to your application's windows and controls. These enablers initialize the Windows specific direct manipulation support.

Note: This requirement does not prevent you from using your own source and target handlers. Just be sure to set the default source and target handler prior to calling the enabling functions.

Public Functions

IDMHandler

Default Handlers

Use these static members to get and set the default source and target handlers used by the various enabling functions.

defaultDragHandler

Returns the default drag handler.

<code>static IDMDragHandler& defaultDragHandler();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

defaultDropTargetHandler

Returns the default drop target handler.

<code>static IDMDropTargetHandler& defaultDropTargetHandler();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

defaultSourceHandler

Returns the default source handler.

Note: The handler returned by this function may have been, or will be, attached to any window for which any of the enabling functions (`enableDragFrom`, `enableDropOn`, or `enableDragDropFor`) have been called. Any changes you make to this handler, such as adding renderers, removing renderers, or changing the renderers effects all of these windows.

<code>static IDMSourceHandler& defaultSourceHandler();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultTargetHandler

Returns the default target handler.

Note: The handler returned by this function may have been, or will be, attached to any window for which any of the enabling functions (`enableDragFrom`, `enableDropOn`, or `enableDragDropFor`) have been called. Any changes you make to this handler, such as adding renderers, removing renderers, or changing the renderers effects all of these windows.

<code>static IDMTargetHandler& defaultTargetHandler();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMHandler

setDefaultSourceHandler

Sets the default source handler to the specified handler. This handler is attached to windows on subsequent calls to `IDMHandler::enableDragFrom` (p. 52) or `IDMHandler::enableDragDropFor` (p. 50).

<code>static void</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setDefaultSourceHandler(IDMSourceHandler& source);</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

source A reference to the source handler object.

setDefaultTargetHandler

Sets the default target handler to the specified handler. This handler is attached to windows on subsequent calls to `IDMHandler::enableDragFrom` (p. 52) or `IDMHandler::enableDragDropFor` (p. 53).

<code>static void</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setDefaultTargetHandler(IDMTargetHandler& target);</code>		<i>Y</i>	<i>Y</i>	<i>N</i>


target A reference to the target handler object.

Enabling Direct Manipulation

Use these static members to provide a simple means of enabling direct manipulation for application windows and controls.

enableDragDropFor

Attaches the default source and target handlers to the specified window and the default source and target renderers to their respective handlers.

	<code>static void</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>enableDragDropFor(IToolBarButton* toolBarButton);</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

toolBarButton

A pointer to the tool bar button object.

This overload of the function is used for tool bar button source and target enablement. A drag item provider is attached to the tool bar button by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the tool bar button, you must attach your own drag item providers.



Attaches the default drag and drop target handlers to the specified window.

IDMHandler

2 static void Win PM Motif
enableDragDropFor(IWindow* window); Y Y N

window A pointer to the window object.

This overload of the function is used for the source and target enablement of controls that are not supported by the default User Interface Class Library support for direct manipulation. These controls can be system-defined controls or custom controls.

Note: You must attach a drag item provider for any types of windows or controls that are not supported by the User Interface Class Library to enable them to support direct manipulation.

Win Attaches the default drag and drop target handlers to the specified window.

3 static void Win PM Motif
enableDragDropFor(IEntryField* entryField); Y Y N

entryField A pointer to the entry field object.

This overload of the function is used for entry field source and target enablement. A drag item provider is attached to the entry field by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the entry field, you must attach your own drag item providers.

Win Attaches the default drag and drop target handlers to the specified window.

4 static void Win PM Motif
enableDragDropFor(IMultiLineEdit* mle); Y Y N

mle A pointer to the multiline edit object.

This overload of the function is used for multiline edit source and target enablement. A drag item provider is attached to the multiline edit (MLE) by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the MLE, you must attach your own drag item providers.

Win Attaches the default drag and drop target handlers to the specified window.

5 static void Win PM Motif
enableDragDropFor(IContainerControl* container); Y Y N

container A pointer to the container object.

This overload of the function is used for container source and target enablement. A drag item provider is attached to the container by default, which provides default

IDMHandler

direct manipulation support for it. If you want specialized direct manipulation support for the container, you must attach your own drag item providers.



Attaches the default drag and drop target handlers to the specified window.

enableDragFrom

Attaches the default source handler to the specified window and the default source renderer to the handler.



```
static void  
enableDragFrom( IEntryField* entryField );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

entryField A pointer to the entry field object.

This overload of the function is used for entry field source enablement. A drag item provider is attached to the entry field by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the entry field, you must attach your own drag item providers.



Attaches the default drag handler to the specified window.



```
static void  
enableDragFrom( IWindow* window );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

window A pointer to the window object.

This overload of the function is used for the source enablement of controls that are not supported by the default User Interface Class Library support for direct manipulation. These controls can be system-defined controls or custom controls.

Note: You must attach a drag item provider for any types of windows or controls that are not supported by the User Interface Class Library to enable them to support direct manipulation.



Attaches the default drag handler to the specified window.



```
static void  
enableDragFrom( IMultiLineEdit* mle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

mle A pointer to the multiline edit object.

This overload of the function is used for multiline edit source enablement. A drag item provider is attached to the multiline edit (MLE) by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the MLE, you must attach your own drag item providers.

IDMHandler

Win Attaches the default drag handler to the specified window.

4 static void enableDragFrom(IContainerControl* container); **Win** **PM** **Motif**
Y Y N

container A pointer to the container object.

This overload of the function is used for container source enablement. A drag item provider is attached to the container by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the container, you must attach your own drag item providers.

Win Attaches the default drag handler to the specified window.

5 static void enableDragFrom(IMenuBar* menuBar); **Win** **PM** **Motif**
I Y N

menuBar A pointer to the menu bar object.

Neither a default source handler nor a drag item provider is attached to the menu bar. The submenus of the menu bar are iterated, and the overload of enableDragFrom is called for each one.

6 static void enableDragFrom(ISubmenu* submenu); **Win** **PM** **Motif**
I Y N

submenu A pointer to the submenu object.

This overload of the function is used for submenu source enablement. A drag item provider is attached to the submenu by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the submenu, you must attach your own drag item providers.

enableDropOn

Attaches the default target handler to the specified window and the default target renderer to the handler.

1 static void enableDropOn(IToolBar* toolBar); **Win** **PM** **Motif**
Y Y I

toolBar A pointer to the tool bar object.

This overload of the function is used for tool bar target enablement. A drag item provider is attached to the tool bar by default, which provides default direct

IDMHandler

manipulation support for it. If you want specialized direct manipulation support for the tool bar, you must attach your own drag item providers.



Attaches the default drop target handler to the specified window.

2

```
static void  
    enableDropOn( IWindow* window );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

window A pointer to the window object.

This overload of the function is used for the target enablement of controls that are not supported by the default User Interface Class Library support for direct manipulation. These controls can be system-defined controls or custom controls.

Note: You must attach a drag item provider for any types of windows or controls that are not supported by the User Interface Class Library to enable them to support direct manipulation.



Attaches the default drop target handler to the specified window.

3

```
static void  
    enableDropOn( IEntryField* entryField );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

entryField A pointer to the entry field object.

This overload of the function is used for entry field target enablement. A drag item provider is attached to the entry field by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the entry field, you must attach your own drag item providers.

4

```
static void  
    enableDropOn( IMultiLineEdit* mle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

mle A pointer to the multiline edit object.

This overload of the function is used for multiline edit target enablement. A drag item provider is attached to the multiline edit (MLE) by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the MLE, you must attach your own drag item providers.



Attaches the default drop target handler to the specified window.

5

```
static void  
    enableDropOn( IContainerControl* container );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMHandler

container A pointer to the container object.

This overload of the function is used for container target enablement. A drag item provider is attached to the container by default, which provides default direct manipulation support for it. If you want specialized direct manipulation support for the container, you must attach your own drag item providers.

Win Attaches the default drop target handler to the specified window.

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class. This class can only be used as the base class for concrete-derived classes, such as IDMSourceHandler (p. 145) and IDMTargetHandler (p. 195).

IDMHandler Provides the default constructor, which takes no arguments.

IDMHandler();

Win
Y

PM
Y

Motif
N

~IDMHandler

virtual
~IDMHandler();

Win
Y

PM
Y

Motif
N

IDMHandler

Handler Processing

Handler-processing members query information about the windows that the handler is attached to.

isContainerControl

Returns true if the window is a container control.

Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>isContainerControl(const IWindow* window) const;</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

Item Providers

These static members set a default item provider for the argument window.

setItemProvider

Sets the default item provider for the argument window.

1	<code>static void setItemProvider(IMultiLineEdit* mle);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

mle A pointer to the multiline edit object.

Sets the default item provider for the multiline edit object.

2	<code>static void setItemProvider(IEntryField* entryField);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

entryField A pointer to the entry field object.

Sets the default item provider for the entry field object.

3	<code>static void setItemProvider(IContainerControl* container);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

container A pointer to the container object.

Sets the default item provider for the container object.

4	<code>static void setItemProvider(ISubmenu* submenu);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>I</i>	<i>Y</i>	<i>N</i>

submenu A pointer to the submenu object.

Sets the default item provider for the submenu object.

IDMHandler

5	<pre>static void setItemProvider(IToolBar* toolBar);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

toolBar A pointer to the tool bar object.

Sets the default item provider for the tool bar object.

6	<pre>static void setItemProvider(IToolBarButton* toolBar);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

toolBar A pointer to the tool bar button object.

Sets the default item provider for the tool bar button object.

Rendering

These members provide a means of accessing the base IDMRenderer (p. 131) objects involved in the direct manipulation described by objects of this class. These objects are used to handle the rendering of drag item objects that are compatible with the supported rendering mechanisms and formats (RMFs).

addRenderer Adds a specified renderer to this handler. The renderer is added to the end of the renderer collection.

Note: Renderers are maintained by position, which are 1-based.

<pre>virtual IDMHandler& addRenderer(const IDMRenderer& newRenderer);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

newRenderer
 A reference to the renderer object.

numberOfRenderers

Returns the number of renderers currently in the renderer collection.

<pre>virtual unsigned numberOfRenderers();</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

removeRenderer

Removes the specified renderer from this handler.

Note: Renderers are maintained by position, which are 1-based.

IDMHandler

1	<code>virtual IDMHandler& removeRenderer(const IDMRenderer& rendererToRemove);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

rendererToRemove

A reference to the renderer object to remove from the renderer collection.

2	<code>virtual IDMHandler& removeRenderer(unsigned position);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

position A value that represents the location of the renderer in the renderer collection.

renderer Returns the renderer at the given position.

Note: Renderers are maintained by position, which are 1-based.

<code>virtual IDMRenderer* renderer(unsigned position);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

replaceRenderer

Replaces a renderer with another specified renderer.

Note: Renderers are maintained by position, which are 1-based.

<code>virtual IDMHandler& replaceRenderer(unsigned position, const IDMRenderer& replacement);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

position A value that represents the location of the renderer to replace in the renderer collection.

replacement

A reference to the replacement renderer object.

setRenderer Sets the renderer for this handler. Removes any pre-existing renderers.

<code>virtual IDMHandler& setRenderer(const IDMRenderer& newRenderer);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

newRenderer

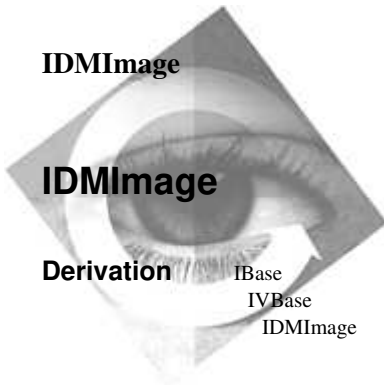
A reference to the renderer object.

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File idmimage.hpp

Members				
Member	Page	Member	Page	
Constructor	61	setBitmap	65	
bitmap	64	setDefaultStyle	67	
bmp	68	setNumberOfPoints	65	
classDefaultStyle	68	setPointArray	65	
closed	68	setPointer	66	
defaultStyle	67	setPointerOffset	66	
noStyle	68	setStretchSize	66	
numberOfPoints	64	setStyle	66	
operator =	64	stretch	68	
pointArray	64	stretchSize	67	
pointer	65	style	67	
pointerOffset	65	transparent	69	
polygon	68	~IDMImage	64	
ptr	68			

The IDMImage class provides the image that represents a drag item during a direct manipulation. The image can be an icon, bitmap, or polygon. The image is displayed while the direct manipulation is in progress.

Create an object of this class and attach it to the drag item that the image represents. Use the member function IDMItem::setImage (p. 77) to attach the drag image object to the drag item. If you do not attach a drag image object to the drag item, an image is created using a default system icon and is attached to the drag item.

Objects of this class have the following attributes in addition to those inherited from its base class:

- An image handle
- A stretch size for the image
- Styles that determine the type and shape of the image
- The offset of the image from the pointing device
- The number of points in an array of points if the style IDMImage::polygon (p. 68) is specified

IDMImage

The drag image styles are defined in `IDM::DragImageStyle` (p. 28). The User Interface Class Library default is `IDM::systemImages`. Call the member function `IDMSourceOperation::setImageStyle` (p. 155) to specify one of the other drag image styles from your implementation of the static member function `IDMItem::generateSourceItems` (p. 97).

Note: If you do not specify one of the other drag image styles, such as `IDM::allStacked`, the image that you construct and attach to the drag item is ignored.



Polygons are not supported.

Public Functions

Constructors

You can construct, copy, assign, and destruct objects of this class.

IDMImage

1	<code>IDMImage(const IDMImage& image);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

image A reference to a drag image object.

Construct objects of this class with this version of the constructor, a copy constructor, if you want to make a copy of a drag image object.

2	<code>IDMImage();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Construct generic objects of this class with this, the default constructor.

3	<code>IDMImage(const IResourceId& resourceId, Boolean iconResource = true, const ISize& pointerOffset = ISize (0 , 0), const ISize& stretchSize = ISize (0 , 0), const Style& style = defaultStyle ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

resourceId A reference to a resource identifier object that wraps the resource ID for the bitmap or icon to load from a resource.

iconResource

A Boolean value that identifies the resource as an icon or a bitmap. If this value identifies the resource as an icon, `IDMImage::ptr` (p. 68) is added to the image's style. Otherwise, the resource is a bitmap and

IDMImage

IDMImage::bmp (p. 68) is added to the image's style. The default identifies the resource as an icon.

pointerOffset

A reference to the pointer offset. Use the pointer offset to set the offset from the pointing device's hot spot to the origin of the drag image. The default sets the origin equal to the pointing device's hot spot.

stretchSize A reference to the stretch size. Use the stretch size to set the stretching dimensions of the drag image. If this parameter is specified, IDMImage::stretch (p. 68) is added to the image's style. If the dimensions are negative, the image is reduced, and if the dimensions are positive, the image is enlarged. The default sets no stretching dimensions.

style A reference to the style object for the drag image. The default style is returned by the static member function IDMImage::defaultStyle (p. 67).

Construct objects of this class with this version of the constructor if you want to use a bitmap or an icon as the drag image and if the bitmap or icon is not already loaded from a resource.

4	IDMImage(const IPointerHandle& icon,		<u>Win</u>	<u>PM</u>	<u>Motif</u>
		const ISize& pointerOffset = ISize (0 , 0),	<i>Y</i>	<i>Y</i>	<i>N</i>
		const ISize& stretchSize = ISize (0 , 0),			
		const Style& style = defaultStyle ());			

icon A reference to the pointer handle that identifies the icon to use as the drag image.

pointerOffset

A reference to the pointer offset. Use the pointer offset to set the offset from the pointing device's hot spot to the origin of the drag image. The default sets the origin equal to the pointing device's hot spot.

stretchSize A reference to the stretch size. Use the stretch size to set the stretching dimensions of the drag image. If this parameter is specified, IDMImage::stretch (p. 68) is added to the image's style. If the dimensions are negative, the image is reduced, and if the dimensions are positive, the image is enlarged. The default sets no stretching dimensions.

style A reference to the style object for the drag image. The default style is returned by the static member function IDMImage::defaultStyle (p. 67).

Construct objects of this class with this version of the constructor if you want to use an icon as the drag image and if the icon is already loaded from a resource.

IDMImage

Adds `IDMImage::ptr` (p. 68) to the image's style.

5	<pre>IDImage(const IBitmapHandle& bitmap, const ISize& pointerOffset = ISize (0 , 0), const ISize& stretchSize = ISize (0 , 0), const Style& style = defaultStyle ());</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

<i>bitmap</i>	A reference to the bitmap handle that identifies the bitmap to use as the drag image.
---------------	---

pointerOffset

A reference to the pointer offset. Use the pointer offset to set the offset from the pointing device's hot spot to the origin of the drag image. The default sets the origin equal to the pointing device's hot spot.

stretchSize A reference to the stretch size. Use the stretch size to set the stretching dimensions of the drag image. If this parameter is specified, `IDMImage::stretch` (p. 68) is added to the image's style. If the dimensions are negative, the image is reduced, and if the dimensions are positive, the image is enlarged. The default sets no stretching dimensions.

style A reference to the style object for the drag image. The default style is returned by the static member function `IDMImage::defaultStyle` (p. 67).

Construct objects of this class with this version of the constructor if you want to use a bitmap as the drag image and if the bitmap is already loaded from a resource.

Adds `IDMImage::bmp` (p. 68) to the image's style.

```

6 IDImage( unsigned long* arrayOfPoints,                               Win PM Motif
           const unsigned long numberOfPoints,                       I Y N
           const ISize& pointerOffset = ISize ( 0 , 0 ),
           const Style& style = defaultStyle ( ) );

```

arrayOfPoints

A pointer to the array of points for the polygon image.

numberOfPoints

A value that identifies the number of points in the array of points.

pointerOffset

A reference to the pointer offset. Use the pointer offset to set the offset from the pointing device's hot spot to the origin of the drag image. The default sets the origin equal to the pointing device's hot spot.

IDMImage

style A reference to the style object for the drag image. The default style is returned by the static member function `IDMImage::defaultStyle` (p. 67).

Construct objects of this class with this version of the constructor if you want to draw a polygon as the drag image.

operator = Assigns a drag image object to another drag image object.

```
IDMImage&
operator =( const IDMImage& image );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

image A reference to a drag image object.

~IDMImage

Deletes the array of points if it exists.

```
virtual
~IDMImage();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Image Processing

Use these members to set and query attributes of this class. These attributes include the following:

- Bitmaps
- Icons
- Polygons
- Style

bitmap Returns the handle of the bitmap image. If the image is not a bitmap, 0 is returned.

```
IBitmapHandle
bitmap() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

numberOfPoints

Returns the number of points in the polygon's array of points.

```
unsigned long
numberOfPoints();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

pointArray Returns a pointer to the array of points if the image style is set to `IDMImage::polygon` (p. 68). Otherwise, 0 is returned.

IDMImage

unsigned long*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pointArray();	<i>I</i>	<i>Y</i>	<i>N</i>

pointer Returns the handle of the icon image. If the image is not an icon, 0 is returned.

IPointerHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pointer() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

pointerOffset Returns the offset from the pointing device's hot spot to the origin of the drag image.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pointerOffset() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setBitmap Sets a bitmap as the drag image and adds IDMImage::bmp (p. 68) to the image's style.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setBitmap(const IBitmapHandle& bitmapImage);	<i>Y</i>	<i>Y</i>	<i>N</i>

bitmapImage

A reference to the bitmap handle that identifies the bitmap to use as the drag image.

setNumberOfPoints

Sets the number of points in the polygon's array of points.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setNumberOfPoints(unsigned long points);	<i>I</i>	<i>Y</i>	<i>N</i>

points A value that identifies the number of points in the array of points.

setPointArray

Sets the array of points for drawing the polygon image and adds IDMImage::polygon (p. 68) to the image's style. If *numberOfPoints* is a value greater than 0, the current array of points is deleted and the new array of points is set. If *numberOfPoints* is 0, the current array of points is deleted and no other processing occurs.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPointArray(unsigned long* arrayOfPoints, unsigned long numberOfPoints);	<i>I</i>	<i>Y</i>	<i>N</i>

arrayOfPoints

A pointer to the array of points for the polygon image.

IDMImage

numberOfPoints

A value that identifies the number of points in the array of points.

setPointer Sets an icon as the drag image and adds IDMImage::ptr (p. 68) to the image's style.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPointer(const IPointerHandle& pointerImage);	<i>Y</i>	<i>Y</i>	<i>N</i>

pointerImage

A reference to the pointer handle that identifies the icon to use as the drag image.

setPointerOffset

Sets the offset from the pointing device's hot spot to the origin of the drag image.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPointerOffset(const ISize& pointerOffset);	<i>Y</i>	<i>Y</i>	<i>N</i>

pointerOffset

A reference to the pointer offset.

setStretchSize

Sets the stretching dimensions of the drag image if the image is a bitmap or icon, and adds IDMImage::stretch (p. 68) to the image's style. If the dimensions are negative, the image is reduced, and if the dimensions are positive, the image is enlarged.

Note: This member function is often required because many bitmaps are too large to use as a drag image. You can use this member function to reduce the size of the bitmap.

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setStretchSize(const ISize& stretchSize);	<i>Y</i>	<i>Y</i>	<i>N</i>

stretchSize A reference to the stretch size.

setStyle

Sets the styles that describe the drag image. The styles are specified in the group IDMImage::Styles (p. 67).

IDMImage&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setStyle(const Style& style);	<i>Y</i>	<i>Y</i>	<i>N</i>

style A reference to the new style object.

IDMImage

stretchSize Returns the stretching dimensions of the drag image. The bitmap or icon is drawn with this size during a direct manipulation.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
stretchSize() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

style Returns the styles that describe the drag image. The styles are specified in the group IDMImage::Styles (p. 67).

Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
style();	<i>Y</i>	<i>Y</i>	<i>N</i>

Styles

Use these members to set and query the styles for this class.

defaultStyle Returns the current default style. This is the same as IDMImage::classDefaultStyle (p. 68) unless IDMImage::setDefaultStyle (p. 67) has been called.

static Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultStyle();	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultStyle

Sets the default style for all subsequent drag images.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultStyle(const Style& style);	<i>Y</i>	<i>Y</i>	<i>N</i>

style A reference to the new style object.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IDMImage

Public Data

Styles

Use these members to set and query the styles for this class.

bmp The drag image is a bitmap.

<code>static const Style bmp;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

classDefaultStyle

Identifies the original default style for this class, which is IDMImage::noStyle (p. 68).

<code>static const Style classDefaultStyle;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

closed If the drag image is a polygon, a closed polygon is formed by moving the current position to the last point in the array of points before drawing the image.

<code>static const Style closed;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

noStyle No styles describe this class. This is the default style setting.

<code>static const Style noStyle;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

polygon The drag image is a pointer to an array of points that is connected to form a polygon. Make the first point of the array (0,0), and place the other points relative to this position.

<code>static const Style polygon;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

ptr The drag image is an icon.

<code>static const Style ptr;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

stretch If the drag image is a bitmap or an icon, the image is stretched to the specified dimensions.

IDMImage

```
static const Style
stretch;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

transparent If the drag image is an icon, an outline of the image is generated and displayed instead of the original image.

```
static const Style
transparent;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IDMImage contains the following nested classes:

IDMImage::Style (see page 70)



IDMImage::Style

IDMImage::Style

Derivation
 IBase
 IBitFlag
 IDMImage::Style

Inherited By None.

Header File idmimage.hpp

The IDMImage::Style class is a nested class that provides static members of the IDMIItem::IDMImage (p. 60) class that define the set of valid drag image styles.

- ptr**
The drag image is an icon.
- bmp**
The drag image is a bitmap.
- polygon**
The drag image is a pointer to an array of points that is connected to form a polygon. Make the first point of the array (0,0), and place the other points relative to this position.
- stretch**
If the drag image is a bitmap or an icon, the image is stretched to the specified dimensions.
- transparent**
If the drag image is an icon, an outline of the image is generated and displayed instead of the original image.
- closed**
If the drag image is a polygon, a closed polygon is formed by moving the current position to the last point in the array of points before drawing the image.
- noStyle**
No styles describe the IDMImage class. This is the default style setting.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IDMImage::Style

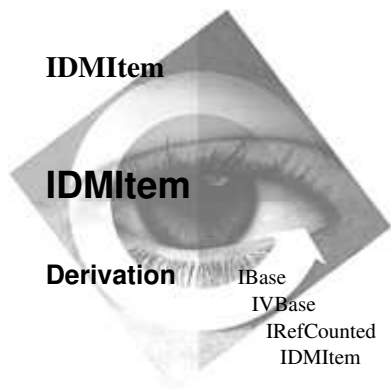
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By

IDMCnrItem	IDMMLItem
IDMEFItem	IDMTBarButtonItem
IDMMenuItem	IDMToolBarItem

Header File idmitem.hpp

Members				
Member	Page	Member	Page	
Constructor	74	moveable	100	
addRMF	90	nativeRF	92	
addType	88	nativeRM	92	
appendRMF	91	nativeRMF	92	
attributes	78	none	99	
canBeCopied	84	object	83	
canBeLinked	84	open	99	
canBeMoved	84	operator =	76	
compressedRMFs	91	prepare	99	
container	98	reference	99	
containerName	78	removableMedia	99	
contents	83	removeRMF	92	
contentsSize	83	removeType	88	
copyable	99	renderer	96	
deleteRMF	91	requiresPreparation	79	
dropStatus	77	rfForThisProcess	93	
enableCopy	84	rfFrom	93	
enableLink	84	rmfFrom	93	
enableMove	85	rmFrom	93	
generateSourceItems	97	rmfs	94	
generateSourceName	98	rmfsFrom	94	
group	98	selectedRMF	96	
hasImage	77	setContainer	79	
hasType	88	setContainerName	80	
image	77	setContents	83	
imageOffset	78	setDropStatus	77	
isContainer	78	setGroup	80	
isGroup	78	setImage	77	
isOnRemovableMedia	79	setNativeRMF	94	
isOpen	79	setObject	84	
isReference	79	setOnRemovableMedia	80	
isTargetTheSource	89	setOpen	80	
linkable	99	setReference	81	
matchingRMFs	92	setRenderer	96	

IDMItem

Member	Page	Member	Page
setRequiresPreparation	81	sourceWindow	82
setRMFs	95	sourceWindowHandle	82
setSelectedRMF	96	strContents	100
setSourceName	81	supportedOperations	82
setSourceWindowHandle	81	supportedOperationsFor	85
setTargetName	82	supportsRMF	95
setTrueType	89	targetDrop	87
setTypes	89	targetEnd	88
sourceDiscard	85	targetName	82
sourceEnd	86	targetOperation	97
sourceItemFor	90	tokenForWPSObject	90
sourceName	82	trueType	89
sourceOperation	96	types	89
sourcePrepare	86	unknown	100
sourcePrint	87	~IDMItem	77
sourceRender	87		

The IDMItem class represents items in a direct manipulation. The items are commonly known as source and target items. Use this class to create source items when starting a direct manipulation on a source window, and target items when dragging over a potential target window.

This class provides virtual functions implementing base support for direct manipulation item objects in the User Interface Class Library. You can create derived classes to support specific items, such as text item objects and container item objects.

Objects of this class have the following attributes in addition to those inherited from its base class:

- A source window handle.
- The types of the dragged object (true and additional).
- The rendering mechanisms and formats (RMFs) of the dragged object (native and additional).
- The container name, which is the source of the direct manipulation operation.
- The source name.
- The suggested target name.
- The item image that visually represents the item while it is being dragged.
- The offset from the pointing device to the origin of the image for this item.
- The source flags providing instructions on how to render an object, such as making the object copyable only or identifying the object as a container of other objects.

IDMItem

- The generic IString buffer contents, which hold a various assortment of bytes that represents the item's data.

IDMSourceRenderer (p. 167) and IDMTargetRenderer (p. 213) objects typically transfer the contents between the source and target windows. The four objects involved in the transfer of the contents (source item, source renderer, target item, and target renderer) must agree on and select a supported RMF.

For example, IDMEFItem (p. 41), IDMSourceRenderer, and IDMTargetRenderer use the item's contents to hold the source entry field text. Derived item classes can use this attribute optionally, provided the renderers they work with use the attribute in a compatible fashion.

- The generic object pointer, which holds a pointer to any type of object.

When you use the library-defined rendering mechanism and the process rendering format, <IDM::rmLibrary,IDM::rfProcess>, the object pointed to is the object being dragged. Therefore, if the target window is in the same process as the source window, the target window can access the dragged object directly.

As with the item's contents, the source items, target items, source renderers, and target renderers must agree on the nature of the value of this field, and derived item classes (and renderer classes) can use this attribute optionally.

- An association with a source or target drag operation object (see IDMSourceOperation (p. 153) or IDMTargetOperation (p. 207)).
- Associations with a selected source or target drag renderer object (see IDMSourceRenderer (p. 167) or IDMTargetRenderer (p. 213)), which the source handler or target handler creates, respectively.



The IDMItem class is a wrapper for the DRAGITEM structure.

Public Functions

Constructors

Use these members to construct, copy, assign, and destruct objects of this class.

IDMItem

A small icon consisting of the letters 'IDM' inside a square border.	IDMItem(IDMSourceOperation* sourceOperation,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	const IString& types,	Y	Y	N
	const unsigned long supportedOperations = unknown, const unsigned long attributes = none);			

IDMItem

sourceOperation

A pointer to the drag source operation object to which this drag item object is to be added.

types A reference to a drag item type or types. Use a comma to separate the individual types if multiple types are identified.

supportedOperations

Identifies the supported operations flags. Operations supported by the User Interface Class Library are IDMItem::copyable (p. 99), IDMItem::linkable (p. 99), and IDMItem::moveable (p. 100). You can also specify user-defined operations.

attributes Identifies item attribute flags. Attributes supported by the User Interface Class Library are IDMItem::none (p. 99), IDMItem::open (p. 99), IDMItem::reference (p. 99), IDMItem::group (p. 98), IDMItem::container (p. 98), IDMItem::prepare (p. 99), and IDMItem::removableMedia (p. 99). You can also specify user-defined attributes.

Use this constructor to construct items for the source of a direct manipulation. It is called by the member function, IDMItem::generateSourceItems (p. 97).

2	IDMItem(IDMTargetOperation* targetOperation, _DRAGITEM* dragItem);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

targetOperation

A pointer to the drag target operation object to which this drag item object is to be added.

dragItem A pointer to the OS/2 Presentation Manager DRAGITEM structure that this item provides a wrapper for.

The User Interface Class Library uses this constructor to construct generic drag items for the target of a direct manipulation. Do not call it.

3	IDMItem(const IDMItem& dragItem);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

dragItem A reference to a drag item object to copy.

Use this constructor, a copy constructor, to make a copy of an existing IDMItem object.

This constructor increments the drag operation's reference count. The drag operation's reference count is used to manage the destruction of the respective source and target operation objects.

IDMItem

4	IDMItem(const Handle& item);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to a handle of a generic drag item that is created by the User Interface Class Library.

Use this constructor to construct drag items for the target of a direct manipulation. It is called by the member function, IDMItemProviderFor::provideTargetItemFor (p. 108), of the drag item provider for the derived item class.

This constructor increments the drag operation's reference count. The drag operation's reference count is used to manage the destruction of the respective source and target operation objects.

5	IDMItem(IDMTargetOperation* targetOperation, IDataObject* dataObject);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

targetOperation A pointer to the drag target operation object to which this drag item object is to be added.

dataObject A pointer to the data object structure that this item provides a wrapper for.

The User Interface Class Library uses this constructor to construct generic drag items for the target of a direct manipulation. Do not call it.

operator = Overloaded definition of the assignment operation to handle assignments of drag items to other drag items.

This operator increments the drag operation's reference count. The drag operation's reference count is used to manage the destruction of the respective source and target operation objects.

1	Handle operator =(const Handle& item);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to a handle of a drag item object to copy.

2	IDMItem& operator =(const IDMItem& item);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to a drag item object to copy.

IDMItem

~IDMItem

This destructor decrements the drag operation's reference count. The drag operation's reference count is used to manage the destruction of the respective source and target operation objects.

```
virtual  
~IDMItem();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drag Image Support

Use these members to set and query the image for the drag item.

hasImage Returns true if the drag item has an associated image.

```
Boolean  
hasImage() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

image Returns a reference to the drag image associated with the drag item. If a drag image is not associated with the drag item, a default drag image is created, associated with the drag item, and returned.

```
virtual IDMImage&  
image();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setImage Sets the item's drag image.

```
virtual IDMItem&  
setImage( IDMImage& image );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

image A reference to the drag image to associate with the drag item.

Drop Status

Use these members to set and query the item's drop status. The drop status determines if this item can be dropped on its current target.

dropStatus Returns the drop status for the drag item.

```
DropIndicator  
dropStatus() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setDropStatus

Sets the drop status for the drag item.

IDMItem

IDMItem& setDropStatus(DropIndicator status);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

status A value that represents the drop status of the drag item.

The drop status values defined by the User Interface Class Library are notOk, ok, operationNotOk, and neverOk. These values are defined in IDM::DropIndicator.

Item Attributes

Use these members to set and query attributes of an item. The attributes include the source and target name, source and target window, and general container information.

attributes Returns the attributes for the drag item. These attributes indicate the nature of the drag item object. Attributes supported by the User Interface Class Library are IDMItem::none (p. 99), IDMItem::open (p. 99), IDMItem::reference (p. 99), IDMItem::group (p. 98), IDMItem::container (p. 98), IDMItem::prepare (p. 99), and IDMItem::removableMedia (p. 99).

virtual unsigned long attributes() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

containerName

Returns the source container's name. If the selected RMF is <IDM::rmFile,IDM::rfText>, this function returns the fully qualified path name where the file resides.

virtual IString containerName() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

imageOffset Returns the offset from the pointing device's hot spot to the origin of the image for the drag item.

virtual ISize imageOffset() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

isContainer Returns true if the drag item is a container of other objects.

virtual Boolean isContainer() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

isGroup Returns true if the drag item is a group of objects.

IDMItem

virtual Boolean
isGroup() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isOnRemovableMedia

Returns true if the drag item is on removable media or may not be easily recovered after a move operation. Examples of removable media are the following:

- Diskettes
- Tapes
- Writable optical disks
- Removable disk drives
- PCMCIA drives

virtual Boolean
isOnRemovableMedia() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isOpen

Returns true if the drag item is open. A drag item could be considered open if it represents the following:

- A file, and the file is currently open
- A program that is currently running
- A device that is currently open

You can interpret these attributes in various ways depending upon your specialized requirements.

virtual Boolean
isOpen() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isReference

Returns true if the drag item is a reference to another object. A drag item could be considered a reference if it is an alias for another object or the original object.

virtual Boolean
isReference() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

requiresPreparation

Returns true if the source requires preparation before rendering. For example, the User Interface Class Library uses this attribute to implement a part of its shared memory RMF support.

virtual Boolean
requiresPreparation() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setContainer Sets or resets the container attribute.

IDMItem

virtual IDMItem& setContainer(Boolean container = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

container A Boolean value that sets or resets the container attribute for the drag item. The default sets the attribute.

setContainerName

Sets the source container name of the item.

IDMItem& setContainerName(const char* containerName);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

containerName
A pointer to a container name. If the selected RMF is <IDM::rmFile,IDM::rfText>, *containerName* identifies the fully qualified path name of the file.

setGroup

Sets or resets the group attribute.

virtual IDMItem& setGroup(Boolean group = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

group A Boolean value that sets or resets the group attribute for the drag item. The default sets the attribute.

setOnRemovableMedia

Sets or resets the removable media attribute. For example, you can set this attribute to true if the object represents a file on a diskette or CD-ROM.

virtual IDMItem& setOnRemovableMedia(Boolean onRemovableMedia = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

onRemovableMedia
A Boolean value that sets or resets the removable media attribute for the drag item. The default sets the attribute.

setOpen

Sets or resets the open attribute. For example, you can set this attribute to true if the object represents a file that is currently open.

virtual IDMItem& setOpen(Boolean open = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

IDMItem

open A Boolean value that sets or resets the open attribute for the drag item.
The default sets the attribute.

setReference Sets or resets the reference attribute. A drag item could be considered a reference if it is an alias for another object or the original object.

```
virtual IDMItem&
    setReference( Boolean reference = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

reference A Boolean value that sets or resets the reference attribute for the drag item. The default sets the attribute.

setRequiresPreparation

Sets or resets the render prepare attribute. For example, the User Interface Class Library uses this attribute to implement a part of its shared memory RMF support.

```
virtual IDMItem&
    setRequiresPreparation( Boolean requiresPrep = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

requiresPrep A Boolean value that sets or resets the render prepare attribute for the drag item. The default sets the attribute.

setSourceName

Sets the source object's name.

```
IDMItem&
    setSourceName( const char* sourceName );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sourceName A pointer to a container name. If the selected RMF is <IDM::rmFile,IDM::rfText>, *sourceName* identifies the file name of the file.

setSourceWindowHandle

Sets the source window handle for the drag item.

```
IDMItem&
    setSourceWindowHandle( IWindowHandle window );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

window The handle of the source window.

IDMItem

setTargetName

Sets the suggested target object's name.

IDMItem& setTargetName(const char* targetName);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

targetName

A pointer to the suggested target object's name. If the selected RMF is <IDM::rmFile,IDM::rfText>, use *targetName* to identify the file name of the target file.

sourceName Returns the source object's name. If the selected RMF is <IDM::rmFile,IDM::rfText>, this function returns the file name.

virtual IString sourceName() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

sourceWindow

Returns a pointer to the source window for the drag item.

virtual IWindow* sourceWindow() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

sourceWindowHandle

Returns the handle of the source window for the drag item.

virtual IWindowHandle sourceWindowHandle() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

supportedOperations

Returns the operations supported by the drag item. Operations supported by the User Interface Class Library are IDMItem::copyable (p. 99), IDMItem::linkable (p. 99), and IDMItem::moveable (p. 100).

virtual unsigned long supportedOperations() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

targetName Returns the suggested name that the drag item object should have at the target. The target might use a different name, for example, if the suggested name is already in use at the target.

IDMItem

```
virtual IString  
    targetName() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Item Contents

Use these members to set and query the contents of the item involved in a direct manipulation.

contents Returns the data for a direct manipulation. Typically, the data is not available at the target prior to the drop. However, if your source and target are within the same process, you can use `IDMItem::sourceItemFor` (p. 90) to obtain the data within your `IDMItemProvider::provideEnterSupport` (p. 103) override, prior to the drop.

```
virtual IString  
    contents() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

contentsSize Returns the length of the data for a direct manipulation. Typically, the data is not available at the target prior to the drop. However, if your source and target are within the same process, you can use `IDMItem::sourceItemFor` (p. 90) to obtain the data within your `IDMItemProvider::provideEnterSupport` (p. 103) override, prior to the drop.

```
virtual unsigned long  
    contentsSize() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

object Returns the pointer to the object of a direct manipulation, such as a container object. Typically, the object is not addressable at the target prior to the drop. However, if your source and target are within the same process, you can use `IDMItem::sourceItemFor` (p. 90) to obtain addressability to the object within your `IDMItemProvider::provideEnterSupport` (p. 103) override, prior to the drop.

```
virtual void*  
    object() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setContents Sets the data within the drag item for a direct manipulation. Use this function to set the data that is associated with the drag item. Typically, the data is not available at the target prior to the drop. However, if your source and target are within the same process, you can use `IDMItem::sourceItemFor` (p. 90) to obtain the data within your `IDMItemProvider::provideEnterSupport` (p. 103) override, prior to the drop.

```
virtual Boolean  
    setContents( const IString& data );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

data A reference to the data that is associated with the drag item.

IDMItem

setObject Sets the pointer to the object of a direct manipulation, such as a container object. Typically, the object is not addressable at the target prior to the drop. However, if your source and target are within the same process, you can use `IDMItem::sourceItemFor` (p. 90) to obtain addressability to the object within your `IDMItemProvider::provideEnterSupport` (p. 103) override, prior to the drop.

<pre>virtual IDMItem& setObject(void* pointerToObject);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

pointerToObject A pointer to the object of the direct manipulation.

Item Operations

Use these members to set and query the operations supported for a drag item object.

canBeCopied Returns true if the drag item can be copied.

<pre>virtual Boolean canBeCopied() const;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

canBeLinked Returns true if the drag item can be linked.

<pre>virtual Boolean canBeLinked() const;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

canBeMoved Returns true if the drag item can be moved.

<pre>virtual Boolean canBeMoved() const;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

enableCopy Enables or disables the copy operation for the drag item.

<pre>virtual IDMItem& enableCopy(Boolean copyable = true);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

copyable A Boolean value that enables or disables the copy operation for the drag item. The default enables the operation.

enableLink Enables or disables the link operation for the drag item.

<pre>virtual IDMItem& enableLink(Boolean linkable = true);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IDMItem

linkable A Boolean value that enables or disables the link operation for the drag item. The default enables the operation.

enableMove Enables or disables the move operation for the drag item.

<pre>virtual IDMItem& enableMove(Boolean moveable = true);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

moveable A Boolean value that enables or disables the move operation for the drag item. The default enables the operation.

supportedOperationsFor

Returns the operations that this item supports using the specified rendering mechanisms and formats (RMFs). This function determines what operation is performed on this item using the selected RMFs. For example, you can designate a printer object as the target and make its default operation a copy rather than a move based upon the RMF of <IDM::rmPrint,IDM::rfUnknown>.

The default implementation of this function calls the member function `IDMItem::supportedOperations` (p. 82) and returns its result.

<pre>virtual unsigned long supportedOperationsFor(const IString& selectedRMFs) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

selectedRMFs
A reference to the selected RMFs for the drag item.

Item Rendering

Use these members for item-specific rendering of the data.

sourceDiscard

Implements item-specific rendering when the operating system issues a discard request. Derived classes must override this function if the item adds the `IDM::rmDiscard` rendering mechanism to its RMFs. This function needs to delete the object that the item represents.

The default implementation of this function does nothing.

<pre>virtual Boolean sourceDiscard(IDMSourceDiscardEvent& event);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>I</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>I</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>I</i>	<i>Y</i>	<i>I</i>					

IDMItem

event A reference to the source discard event.



The discard request is issued by the Workplace Shell when the item is dropped on a shredder object.

sourceEnd Implements item-specific rendering when source or target rendering of the item has finished. This function is called when the target notifies the source that it has finished rendering the item.

Derived classes should override this function to accomplish any of the following:

- Perform any special cleanup before the direct manipulation ends
- Fulfill other requirements imposed by a nondefault renderer

The default implementation of this function does nothing.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceEnd(IDMSourceEndEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source end event.

sourcePrepare

Implements item-specific rendering when source rendering is used to render the item. Override this function in your derived item class if you use a nondefault renderer that supports source rendering and the renderer requires item-specific processing, and if source preparation is a required part of the rendering.

Derived classes should override this function to accomplish any of the following:

- To creation a secondary thread to perform the source rendering.
- To determine the size of a shared memory buffer. Do this if you want to implement your own shared memory RMF.
- To fulfill other requirements imposed by the nondefault renderer.

The default implementation of this function does nothing.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourcePrepare(IDMSourcePrepareEvent& event);	<i>I</i>	<i>Y</i>	<i>I</i>

event A reference to the source prepare event.

IDMItem

sourcePrint Implements item-specific rendering when the operating system issues a print request. Derived classes must override this function if the item adds the IDM::rmPrint rendering mechanism to its RMFs. This function needs to print the object that the item represents.

The default implementation of this function does nothing.

```
virtual Boolean  
    sourcePrint( IDMSourcePrintEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>I</i>

event A reference to the source print event.

PM The print request is issued by the Workplace Shell when the item is dropped on a printer object.

sourceRender

Implements item-specific rendering when source rendering is used to render the item. Override this function in your derived item class if you use a nondefault renderer that supports source rendering and the renderer requires item specific processing.

The default implementation of this function does nothing.

```
virtual Boolean  
    sourceRender( IDMSourceRenderEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

event A reference to the source render event.

targetDrop Implements item-specific rendering when the item is dropped on a target. Derived classes must override this function to process the data that is transferred to the target window as a result of target rendering.

If source rendering is not used, or if source rendering is performed but not in a secondary thread, target cleanup should occur in this function.

The default implementation of this function does nothing.

Note: If multiple items are dropped, this function is called once for each item. The items are processed in the reverse order in which they were added to the source operation in the member function, IDMItem::generateSourceItems (p. 97).

```
virtual Boolean  
    targetDrop( IDMTargetDropEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target drop event.

IDMItem

targetEnd

Implements item-specific rendering when the item is dropped on a target. Derived classes must override this function when a nondefault source renderer renders the data on a secondary thread. The override must process the data that is transferred to the target window as a result of the source rendering. Target cleanup should also occur in the override.

The source renderer informs the target that it has finished by calling `IDMSourceRenderer::informTargetOfCompletion` (p. 169). The call to this function is unique because the source actually posts an event to inform the target that it is finished; thus, this function is called asynchronously.

Ensure that there is no overlap between this function or `IDMItem::targetDrop` (p. 87), or the results may be unpredictable.

The default implementation of this function does nothing.

Note: If multiple items are dropped, this function is called once for each item.

<code>virtual Boolean</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>targetEnd(IDMTargetEndEvent& event);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>
<i>event</i>	A reference to the target end event.		

Item Types

Use these members to set and query the drag item's types. Use a type or types to uniquely identify drag items that represent objects, such as container objects and bitmaps.

addType

Adds one or more additional types to the drag item. Use this function in a derived class' constructor to add the type or types supported by the derived class.

<code>virtual IDMItem&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>addType(const char* aType);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>
<i>aType</i>	A pointer to the new type to add.		

hasType

Returns true if the drag item supports the specified type.

<code>virtual Boolean</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>hasType(const char* aType) const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>
<i>aType</i>	A pointer to the type to query.		

removeType Removes the specified type from the drag item.

IDMItem

```
virtual IDMItem&
    removeType( const char* aType );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

aType A pointer to the type to remove.

setTrueType Sets the drag item's true type.

```
virtual IDMItem&
    setTrueType( const char* aType );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

aType A pointer to the true type.

setTypes Sets the drag item's types. The following line of code removes all of the types for the drag item:

```
setTypes("");
```

```
virtual IDMItem&
    setTypes( const char* types );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

types A pointer to the new types.

trueType Returns the true type of the drag item. For example, if the object is a text file, the type is IDM::plainText. If the file is an icon, the type is IDM::icon.

```
virtual IString
    trueType() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

types Returns all of the drag item's supported types. Individual types are separated by a comma. IDM::Type (p. 31) contains a list of the User Interface Class Library's defined types.

```
virtual IString
    types() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Item Utilities

Use these members to provide miscellaneous utility services for a drag item.

isTargetTheSource

Returns true if source and target are the same window.

IDMItem

Boolean

isTargetTheSource() const;

Win

PM

Motif

Y

Y

N

sourceItemFor

Retrieves the source item's handle based upon the handle of the target item. The source and target must be in the same process. Otherwise, 0 is returned.

Use this function to access the source item. You can obtain the source item's data via IDMItem::contents (p. 83), and you can obtain addressability to the associated object via IDMItem::object (p. 83).

static Handle

sourceItemFor(const Handle& targetItem);

Win

PM

Motif

Y

Y

N

targetItem A reference to the handle of a target drag item. If the drag item is not a target drag item, 0 is returned.

Exceptions	
IAccessError	The source item handle could not be retrieved. The query of the process and thread identifier failed. The window handle may have been invalid.

tokenForWPSObject

Returns the token for the Workplace Shell object. Use the OBJECT_FOR_PREC macro to convert the token into a WPObject pointer.

Note: The application using the token must be shell-enabled or OBJECT_FOR_PREC traps.

unsigned long

tokenForWPSObject() const;

Win

PM

Motif

N

Y

N

Rendering Mechanisms and Formats

Use these members to create, delete, manipulate, set, and query the rendering mechanisms and formats (RMFs) for a drag item.

addRMF Adds one or more RMFs to the drag item.



virtual IDMItem&

addRMF(const IString& rmf);

Win

PM

Motif

Y

Y

N

IDMItem

rmf A reference to the RMFs string.

The RMFs string must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

2	<pre>virtual IDMItem& addRMF(const IString& rm, const IString& rf);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------	---	-----------------	----------------	-------------------

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

The RMFs string that is added is created from separate rendering mechanism and rendering format strings. Both strings can have multiple values that must be separated with commas.

appendRMF Appends the RMF string, *rmf*, to the RMFs string, *rmfs*, and separates the two strings by a comma. Both strings must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

<pre>static void appendRMF(IString& rmfs, const IString& rmf);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

rmfs A reference to the RMFs string that is the target of the append.

rmf A reference to the RMF string to append.

compressedRMFs

Returns a compressed version of the RMFs string with all possible cross products converted to cross-product form. The cross-product form is "(rm)x(rf1,rf2)", where *rm* represents the rendering mechanism and *rf1* and *rf2* represent the rendering formats.

<pre>static IString compressedRMFs(const IString& rmfs);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

rmfs A reference to the RMFs string to compress.

deleteRMF Deletes the RMF string, *rmf*, from the RMFs string, *rmfs*. Both strings must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

<pre>static void deleteRMF(IString& rmfs, const IString& rmf);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

IDMItem

rmfs A reference to the RMFs string that is the target of the delete.
rmf A reference to the RMF string to delete.

matchingRMFs

Returns the intersection of the two RMF strings. Both strings must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

static IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
matchingRMFs(const IString& rmfs1,	<i>Y</i>	<i>Y</i>	<i>N</i>
const IString& rmfs2,			
Boolean firstOnly = false);			

rmfs1 A reference to the first RMFs string.
rmfs2 A reference to the second RMFs string.
firstOnly A Boolean value that indicates that only the first matching RMF is returned. The default value is for all matching RMFs to be returned.

nativeRF Returns the native rendering format of the drag item.

virtual IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
nativeRF() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

nativeRM Returns the native rendering mechanism of the drag item.

virtual IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
nativeRM() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

nativeRMF Returns the native RMF of the drag item.

virtual IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
nativeRMF() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

removeRMF Removes an RMF from the drag item.

 virtual IDMItem&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
removeRMF(const IString& rmf);	<i>Y</i>	<i>Y</i>	<i>N</i>

rmf A reference to the RMF string.

The RMF string must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

IDMItem

2	virtual IDMItem& removeRMF(const IString& rm, const IString& rf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

rm and *rf* must represent a single rendering mechanism and a single rendering format, respectively. Multiple rendering mechanisms or rendering formats are not supported.

rfForThisProcess

Returns the string, DRF_PROCESS=pid, where *pid* is the ASCII representation of the process identifier (ID). Use this function to create the process rendering format.

	static IString rfForThisProcess();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	---------------------------------------	-------------------------------	------------------------------	---------------------------------

rfFrom

Returns the rendering format from the specified RMF string. The RMF string must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

	static IString rfFrom(const IString& rmf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	---	-------------------------------	------------------------------	---------------------------------

rmf A reference to the RMF string.

rmfFrom

Creates a rendering mechanism and format (RMF)-ordered pair: "<rm,rf>". *rm* and *rf* must represent a single rendering mechanism and a single rendering format, respectively. Multiple rendering mechanisms or rendering formats are not supported.

	static IString rmfFrom(const IString& rm, const IString& rf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	---	-------------------------------	------------------------------	---------------------------------

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

rmFrom

Returns the rendering mechanism from the specified RMF string. The RMF string must be in either the ordered-pair format, "<rm,rf>", or cross-product format, "(rm)x(rf1,rf2)".

	static IString rmFrom(const IString& rmf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	---	-------------------------------	------------------------------	---------------------------------

rmf A reference to the RMF string.

IDMItem

rmfs Returns all RMFs for the drag item.

<pre>virtual IString rmfs() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

rmfsFrom Returns an RMFs string.

1 <pre>static IString rmfsFrom(const IString& rmfs);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

rmfs A reference to the RMFs string.

The RMFs string is created from the specified RMFs string. Use this version of the function to expand a cross-product form into ordered pairs.

The following line of code illustrates a cross-product form:

```
IDMItem::rmfsFrom( "(rm1)x(rf1,rf2)" );
```

The RMFs string returned from the sample line of code is "<rm1,rf1>,<rm1,rf2>".

2 <pre>static IString rmfsFrom(const IString& rm, const IString& rf);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

The RMFs string is created from separate rendering mechanism and rendering format strings. Both strings can have multiple values that must be separated with commas.

The RMFs string is returned in the ordered-pair format.

The following line of code illustrates passing multiple values:

```
IDMItem::rmfsFrom( "rm1,rm2", "rf1,rf2" );
```

The RMFs string returned from the sample line of code in the ordered-pair format is "<rm1,rf1>,<rm1,rf2>,<rm2,rf1>,<rm2,rf2>".

setNativeRMF

Sets the native RMF to the specified value.

1 <pre>virtual IDMItem& setNativeRMF(const IString& rmf);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IDMItem

rmf A reference to the RMF string.

rmf must represent a single RMF in the ordered-pair format, "<rm,rf>". Multiple RMFs are not supported.

2	virtual IDMItem& setNativeRMF(const IString& rm, const IString& rf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	-------------------------------	------------------------------	---------------------------------

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

rm and *rf* must represent a single rendering mechanism and a single rendering format, respectively. Multiple rendering mechanisms or rendering formats are not supported.

setRMFs

Sets the new RMFs for the drag item. Use the following line of code to remove all of the RMFs for the item:

```
setRMFs( "" );
```

virtual IDMItem& setRMFs(const IString& rmfs);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

rmfs A reference to the new RMFs string.

supportsRMF

Returns true if the item supports the specified RMFs.

1	virtual Boolean supportsRMF(const IString& rm, const IString& rf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	-------------------------------	------------------------------	---------------------------------

rm A reference to the rendering mechanism string.

rf A reference to the rendering format string.

Both strings can have multiple values that must be separated with commas.

2	virtual Boolean supportsRMF(const IString& rmf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

rmf A reference to the RMFs string.

The RMFs string can be either in the cross-product format, "(rm)x(rf1,rf2)" or the ordered-pair format, "<rm1,rf1>,<rm2,rf2>".

IDMItem

Selected Renderer

Use these members to set and query the renderer that is selected for the drag item.

renderer Returns the position of the selected renderer for this drag item.

unsigned renderer() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
-------------------------------	------------------------	-----------------------	--------------------------

selectedRMF Returns the selected rendering mechanism and format (RMF) for this item in an ordered-pair format, such as: "<IDM::rmLibrary,IDM::rfText>".

IString selectedRMF() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---------------------------------	------------------------	-----------------------	--------------------------

setRenderer Sets the selected renderer for the drag item. The function's argument is the relative position of the specified renderer within the source or target handler's collection of renderers.

Note: This function should only be called by the source or target handler.

IDMItem& setRenderer(unsigned position);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

position A value that represents the renderer's position.

setSelectedRMF

Sets the selected rendering mechanism and format (RMF) for this drag item. The function's argument must be an ordered-pair format, such as: "<IDM::rmLibrary,IDM::rfText>".

IDMItem& setSelectedRMF(const IString& rmf);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

rmf A reference to the RMF string.

Source and Target Operation

Use these members to access the source and target operation objects.

sourceOperation

Returns a pointer to the source operation for the drag item. If the item represents a target item, 0 is returned.

IDMItem

```
virtual IDMSourceOperation*  
    sourceOperation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

targetOperation

Returns a pointer to the target operation for the drag item. If the item represents a source item, 0 is returned.

```
virtual IDMTargetOperation*  
    targetOperation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

generateSourceItems

Generates generic drag items. This function is called by the implementation of `IDMItemProvider::provideSourceItems` (p. 105) in the template class `IDMItemProviderFor` (p. 107). This arrangement is designed to permit you to implement all of your application-specific behavior in a single class derived from `IDMItem` or one of its derivatives. The corresponding drag item provider object is constructed using the `IDMItemProviderFor` template with your derived item class.

The default implementation of this function in this class creates generic `IDMItem` objects and adds them to the source operation. If you use the `IDMItemProviderFor` template class and you have defined a derived item class, you must implement `IDMItem::generateSourceItems` in your derived item class. In your implementation, create one or more objects of your derived item class and call `IDMOperation::addItem` (p. 123) to add the objects to the source operation.

Note: Use the *sourceOperation* parameter to access `IDMOperation::addItem`.

```
static Boolean  
    generateSourceItems( IDMSourceOperation* sourceOperation );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sourceOperation

A pointer to the source operation object.

Inherited Public Functions

IRefCounted		
addRef	removeRef	useCount

IDMItem

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Source Name

These members generate the source name when the IDM::rfText-rendering format is used.

generateSourceName

Use this function to generate the source name for the implementation of the IDM::rfText-rendering format.

QString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
generateSourceName();	<i>Y</i>	<i>Y</i>	<i>N</i>

Public Data

Attribute Flags

Use these static members to define the drag item's attribute flags.

Note: These static members represent *bit masks*, which are patterns of characters used to control portions of another pattern of characters. Any user-defined values must be greater than the IDMItem::removableMedia (p. 99) value.

container The source object is a container of other objects.

static const unsigned long container;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

group The source object is a group of objects.

static const unsigned long group;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMItem

none No attributes are defined for the source object.

static const unsigned long
none;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

open The source object is open.

static const unsigned long
open;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

prepare The source object requires preparation before it establishes a data transfer conversation.

static const unsigned long
prepare;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

reference The source object is a reference to another object.

static const unsigned long
reference;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

removableMedia

The source object is on removable media, or the source object cannot be recovered after a move operation.

static const unsigned long
removableMedia;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Supported Operations Flags

Use these static members to define the supported operations of the drag item.

Note: These static members represent *bit masks*, patterns of characters used to control portions of another pattern of characters. Any user-defined values must be greater than the IDMItem::linkable value.

copyable The source object can be copied to the specified target.

static const unsigned long
copyable;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

linkable The source object can be linked to the specified target.

IDMItem

static const unsigned long
linkable;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

moveable The source object can be moved to the specified target.

static const unsigned long
moveable;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

unknown No supported drag operations are available.

static const unsigned long
unknown;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Protected Data

Item Data

These members store data for the drag item.

strContents Holds data (contents) of the drag item. This data member is protected to allow derived classes to either return the contents of it via IDMItem::contents (p. 83) or to allow IDMItem::object (p. 83) to return a pointer to it per a derived item implementation.

IString
strContents;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Type Definitions

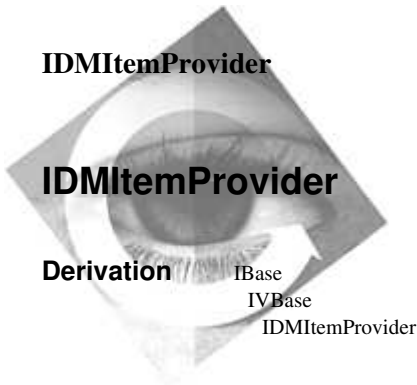
Handle typedef IReference < IDMItem > Handle;

IDMItem::Handle provides access to the IDMItem objects associated with a direct manipulation. The handle manages the references to the IDMItem object and ensures that the object is not destructed until the direct manipulation is completed.

IDMItem

Use Handle to reference an item handle within this class and IDMItem::Handle externally.

The handle provides a pointer operator, which enables the handle to be treated just like a pointer to an IDMItem object.



Inherited By IDMIItemProviderFor

Header File idmprov.hpp

Members		Member	Page	Member	Page
	Constructor	103		provideSourceItems	105
	provideEnterSupport	103		provideTargetItemFor	105
	provideHelpFor	104		~IDMIItemProvider	103
	provideLeaveSupport	105			

The IDMIItemProvider class provides application-specific direct manipulation behavior for windows. Construct objects from a derived class or the IDMIItemProviderFor (p. 107) template class and attach them to source and target windows that you enable for direct manipulation. Attach the objects with the member function IWindow::setItemProvider (Vol. II).

The responsibilities of this class include the following:

- Providing source and target items (as described)
- Providing help when requested while a drag item is positioned over a target
- Providing application-defined checking when a drag item is entering or moving over a target
- Providing application-defined cleanup when a drag item is exiting a target
- Providing other application-defined support; for example, drawing the target emphasis

When the source handler detects that a direct manipulation is started in one of the windows it is attached to, it calls IDMIItemProvider::provideSourceItems (p. 105).

When drag items enter a potential target, generic IDMIItem (p. 72) objects are created to represent them. Next, the target handler calls IDMIItemProvider::provideTargetItemFor (p. 105) once for each drag item. This gives a derived IDMIItemProvider class the opportunity to replace the generic IDMIItem

IDMItemProvider

objects with objects of some IDMItem-derived class that provides application-specific behavior.

Note: See the direct manipulation section of the *User's Guide* for more information on how to construct derived IDMItemProvider classes. Also, see the template class IDMItemProviderFor (p. 107), which you can use to construct derived IDMItemProviderFor classes for your IDMItem derived classes.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMItemProvider

Provides the default constructor to construct objects of this class.

IDMItemProvider();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

~IDMItemProvider

virtual ~IDMItemProvider();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

Drag Item Support

Use these members to provide event-specific support during a direct manipulation.

provideEnterSupport

Provides support for the target enter event. This function is called when the user moves a drag item over the target for which the provider is providing direct manipulation support.

Derived classes should override this function to accomplish any of the following:

- To draw target emphasis on the target window.
- To validate that the drag item can be dropped. If it cannot be dropped, call IDMTargetEnterEvent::setDropIndicator (p. 190) to set one of the drop indicator values defined in IDM::DropIndicator (p. 29).

The default implementation of this function does the following:

IDMItemProvider

- For multiline edit (MLE) controls and the entry field, if the source and target windows are the same window, the drop indicator is set to `IDM::neverOk`.
- For containers, if the potential target is a container object and if the drag item is directly over the object, you can validate that the container object can be dropped on via the member function `IContainerObject::isDropOnAble` (Vol. III). If the drag item is between container objects in one of the container list views, such as the details view, this test is not performed. `IDMTargetEnterEvent::isDragAfter` (p. 188) is used to determine if the drag item is between container objects if the objects are displayed in one of the container list views.
- For intraprocess containers, if the operation is a move, `IContainerControl::isMoveValid` (Vol. III) is called to validate the move.
- For tool bar buttons, if the source and target buttons are the same button, if the button already exists on the tool bar, or if one of the drag item types is not `IDM::toolBarButton`, `IDM::menuItem`, or `IDM::bitmap`, the drop indicator is set to `IDM::neverOk`.
- For tool bars, if the button already exists on the tool bar, or if one of the drag item types is not `IDM::toolBarButton` or `IDM::menuItem`, the drop indicator is set to `IDM::neverOk`.

```
virtual Boolean  
    provideEnterSupport( IDMTargetEnterEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target enter event.

provideHelpFor

Provides target-specific drop help. This function is called when the user requests help while a drag item is positioned over a potential target.

Derived classes should override this function to provide the specific help for the target of a drop. For example, you can provide help that details the processing that occurs if the drag item is dropped on the target.

The default implementation of this function does nothing.

```
virtual Boolean  
    provideHelpFor( IDMTargetHelpEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>N</i>

event A reference to the target help event.

IDMItemProvider

provideLeaveSupport

Provides support for the target leave event. This function is called when the user moves a drag item off of the target for which the provider is providing direct manipulation support.

Derived classes should override this function to perform any required cleanup or to remove target emphasis from the target.

The default implementation of this function does nothing.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
provideLeaveSupport(IDMTargetLeaveEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target leave event.

provideSourceItems

Adds IDMItem (p. 72) or IDMItem-derived class objects to the specified *sourceOperation*. This function is called when a direct manipulation starts on a source window.

If drag items are provided, true is returned.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
provideSourceItems(<i>Y</i>	<i>Y</i>	<i>N</i>
const IDMSourceOperation::Handle& sourceOperation);			

sourceOperation

A reference to the handle of the source operation.

provideTargetItemFor

Replaces generic IDMItem (p. 72) objects with an IDMItem-derived class during the processing of the initial target enter event.

The default implementation returns the item handle, *dragItem*, that was passed to the function.

virtual IDMItem::Handle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
provideTargetItemFor(const IDMItem::Handle& dragItem);	<i>Y</i>	<i>Y</i>	<i>N</i>

dragItem A reference to the handle of the generic drag item.

IDMItemProvider

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMItemProviderFor

Derivation

```

IBase
  IVBase
    IDMItemProvider
      IDMItemProviderFor
  
```

Inherited By None.

Header File idmprov.hpp

Members				
Member	Page	Member	Page	
Constructor	108	provideTargetItemFor	108	
provideSourceItems	108	~IDMItemProviderFor	108	

The IDMItemProviderFor class is a template class that generates an IDMItemProvider-derived class for your derived IDMItem (p. 72) classes.

This class overrides the key functions of the IDMItemProvider (p. 102) class and dispatches static functions of your IDMItem-derived class specified as the template argument. IDMItemProviderFor::provideSourceItems calls the static IDMItem::generateSourceItems (p. 97) member function of your derived IDMItem class.

IDMItemProvider::provideTargetItemFor creates a new object of your IDMItem-derived class. This requires that your IDMItem-derived class provide a constructor that accepts an argument of the type *const IDMItem::Handle&*. This argument is a reference to the handle of the generic IDMItem object.

Note: If you need to override additional IDMItemProvider functions (such as provideHelpFor or provideEnterSupport), you can derive your item provider class from this template class.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMItemProviderFor

IDMItemProviderFor

Provides the default constructor to construct objects of this class.

```
IDMItemProviderFor();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IDMItemProviderFor

```
virtual  
~IDMItemProviderFor();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drag Item Support

Use these members to provide event-specific support during a direct manipulation. These members are called when a direct manipulation starts on a source window and during the processing of the initial target enter event.

provideSourceItems

Adds IDMItem (p. 72) or IDMItem-derived class objects to the specified *sourceOperation*. This function is called when a direct manipulation starts on a source window.

This function delegates requests to the static member function IDMItem::generateSourceItems (p. 97) of the template's argument class, <T>, where *T* represents the derived IDMItem class. <T>::generateSourceItems creates appropriate source drag items and adds them to the specified source operation object

If source drag items are provided, true is returned.

```
virtual Boolean  
provideSourceItems(  
    const IDMSourceOperation::Handle& sourceOperation);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sourceOperation

A reference to the handle of the source operation.

provideTargetItemFor

Replaces generic IDMItem (p. 72) objects with an IDMItem-derived class during the processing of the initial target enter event.

This function creates a new object of the template's argument class, <T>, where *T* represents the derived IDMItem class, and replaces the specified generic IDMItem object, *dragItem*, in the target operation with the new object. The template argument

IDMItemProviderFor

class must provide a constructor that accepts an argument of the type *const IDMItem::Handle&*. This argument is a reference to the handle of the generic IDMItem object.

```
virtual IDMItem::Handle  
    provideTargetItemFor( const IDMItem::Handle& dragItem );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

dragItem A reference to the handle of the generic drag item.

Inherited Public Functions

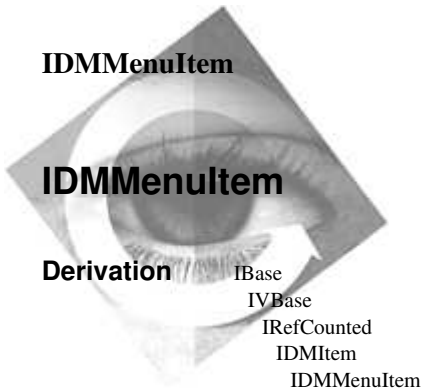
IDMItemProvider		
provideEnterSupport	provideHelpFor	provideLeaveSupport

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File idmmenit.hpp

Members	Member	Page	Member	Page
	Constructor	110	supportedOperationsFor	112
	generateSourceItems	112	~IDMMenuItem	111
	sourceEnd	111		

The IDMMenuItem class represents menu-specific items in a direct manipulation. Menus create objects of the IDMMenuItem class when a direct manipulation begins in a source menu.

This class provides virtual functions that implement direct manipulation support for menus. Derive item classes to support the direct manipulation of menu drag items when the default User Interface Class Library support does not meet your requirements.


Note: This class does not support menus as the target of a direct manipulation.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMMenuItem

	IDMMenuItem(IDMSourceOperation* sourceOperation);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>I</i>	<i>Y</i>	<i>N</i>

IDMMenuItem

sourceOperation

A pointer to the drag source operation object to which this menu drag item object is to be added.

Use this constructor to construct menu drag items for the source of a direct manipulation. It is called by the member function, `IDMMenuItem::generateSourceItems` (p. 112).

This constructor determines which menu item is directly under the pointing device when the direct manipulation begins and selects the menu item as a visual cue. It also disables the dismissing of the menu until the direct manipulation is complete.

2	<code>IDMMenuItem(const IDMItem::Handle& dragItem);</code>	<u>Win</u> <u>PM</u> <u>Motif</u>
		<i>I</i> <i>Y</i> <i>N</i>

dragItem A reference to a handle of a generic drag item that is created by the User Interface Class Library.

This constructor is not used. Its definition is necessary to satisfy the requirements of the `IDMItemProviderFor` (p. 107) template class.

~IDMMenuItem

Dismisses the menu if it has not been dismissed, because one of the following occurs:

- The direct manipulation is cancelled.
- Help is requested.
- The drop is not allowed.

<code>virtual</code> <code> ~IDMMenuItem();</code>	<u>Win</u> <u>PM</u> <u>Motif</u>
	<i>I</i> <i>Y</i> <i>N</i>

Menu Cleanup

Use these members to handle cleanup of the menu after the direct manipulation has completed.

sourceEnd Processes a source end event for the menu drag item. This function dismisses the menu that is the source of the direct manipulation.

<code>virtual Boolean</code> <code> sourceEnd(IDMSourceEndEvent& event);</code>	<u>Win</u> <u>PM</u> <u>Motif</u>
	<i>I</i> <i>Y</i> <i>N</i>

event A reference to the source end event.

IDMMenuItem

Menu Item Operations

Use these members to set the supported direct manipulation operations for this class.

supportedOperationsFor

Returns IDMItem::linkable (p. 99) as the only operation the menu drag item supports.

```
virtual unsigned long  
supportedOperationsFor( const IString& selectedRMFs ) const; Win PM Motif  
                                                           I   Y   N  
  
selectedRMFs
```

A reference to the selected RMFs for the drag item.

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

generateSourceItems

Generates a menu drag item that represents a menu item object. The menu item object is selected in the source menu.

This function is called by the menu's implementation of
IDMItemProvider::provideSourceItems (p. 105) in the template class
IDMItemProviderFor (p. 107).

The default implementation of this function in this class creates IDMMenuItem objects and adds them to the source operation. If you use the IDMItemProviderFor template class and you have defined a derived item class, you must implement IDMMenuItem::generateSourceItems in your derived menu drag item class. In your implementation, create an object of your derived menu item class and call IDMOperation::addItem (p. 123) to add the objects to the source operation.

The default drag operation for menu drag items is IDMItem::linkable (p. 99).

Note: Use the *sourceOperation* parameter to access IDMOperation::addItem.

```
static Boolean  
generateSourceItems( IDMSourceOperation* sourceOperation ); Win PM Motif  
                                                           I   Y   N  
  
sourceOperation
```

A pointer to the source operation object.

Inherited Public Functions

IDMItem		
addRMF	nativeRF	setRequiresPreparation
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IRefCounted		
addRef	removeRef	useCount

IDMMenuItem

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IBase		
recoverable	unrecoverable	



IDMMLEItem

Derivation

```

IBase
  IVBase
    IRefCounted
      IDMItem
        IDMMLEItem
  
```

Inherited By None.

Header File idmmleitem.hpp

Members	Member	Page	Member	Page
	Constructor	115	sourceEnd	116
	generateSourceItems	117	targetDrop	117
	object	117	~IDMMLEItem	116

The IDMMLEItem class represents multiline edit (MLE)-specific items in a direct manipulation. MLEs create objects of the IDMMLEItem class when either of the following occurs:

- A direct manipulation begins in a source MLE
- An MLE item is dragged over a potential target MLE during a direct manipulation

In addition to attributes inherited from IDMItem, objects of this class have a pointer to the text object that the MLE item represents.

This class provides virtual functions that implement direct manipulation support for the MLE. Derive item classes to support the direct manipulation of MLE items when the default User Interface Class Library support does not meet your requirements.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMMLEItem

IDMMLEItem

1	IDMMLEItem(IDMSourceOperation* sourceOperation);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

sourceOperation

A pointer to the drag source operation object to which this MLE item object is to be added.

Use this constructor to construct MLE items for the source of a direct manipulation. It is called by the member function, IDMMLEItem::generateSourceItems (p. 117).

Note: This constructor sets the IDM::rmLibrary rendering mechanism and the IDM::rfProcess rendering format. If the text size is less than or equal to 255 bytes and the IDM::rmFile rendering mechanism has not been added in a derived class, the IDM::rfText rendering format is added. Otherwise, the IDM::rfSharedMem rendering format is added.

2	IDMMLEItem(const IDItem::Handle& dragItem);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

dragItem A reference to a handle of a generic drag item that is created by the User Interface Class Library.

Use this constructor to construct MLE items for the target of a direct manipulation. It is called by the member function, IDItemProviderFor::provideTargetItemFor (p. 108), of the MLE's item provider for the MLE item class.

~IDMMLEItem

virtual		<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IDMMLEItem();		<i>Y</i>	<i>Y</i>	<i>N</i>

Drop Processing

Use these members during the drop operation of a direct manipulation.

sourceEnd Processes a source end event for the MLE item. This function removes the selected text from the source MLE during a move operation.

virtual Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceEnd(IDMSourceEndEvent& event);		<i>Y</i>	<i>N</i>	<i>N</i>

event A reference to the source end event.

IDMMLEItem

targetDrop Implements MLE item-specific rendering when the MLE item that represents a text object is dropped on a target MLE. This function sets the text into the target MLE. Derived classes should override this function to set the associated text object into the target MLE when the default support does not meet your requirements.

```
virtual Boolean  
    targetDrop( IDMTargetDropEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target drop event.

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

generateSourceItems

Generates MLE drag items that represent text objects. The text object can be the entire contents of the source MLE or a selected portion.

This function is called by the MLE's implementation of IDMItemProvider::provideSourceItems (p. 105) in the template class IDMMItemProviderFor (p. 107).

The default implementation of this function in this class creates IDMMLEItem objects and adds them to the source operation. If you use the IDMMItemProviderFor template class and you have defined a derived item class, you must implement IDMMLEItem::generateSourceItems in your derived MLE item class. In your implementation, create an object of your derived MLE item class and call IDMOperation::addItem (p. 123) to add the objects to the source operation.

Note: Use the *sourceOperation* parameter to access IDMOperation::addItem.

```
static Boolean  
    generateSourceItems( IDMSourceOperation* sourceOperation );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

sourceOperation
A pointer to the source operation object.

object Returns a pointer to the contents of the IString that contains either all of or the selected portion of the source MLE's text. The pointer can be cast to the type *char ** to directly access the text.

```
virtual void*  
    object() const;
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMMLEItem

Inherited Public Functions

IDMItem		
addRMF	nativeRF	setRequiresPreparation
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IRefCounted		
addRef	removeRef	useCount

IDMMLEItem

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

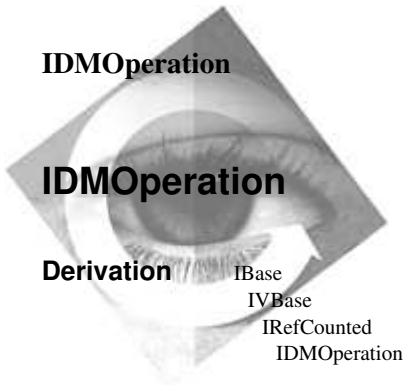
Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IBase		
recoverable	unrecoverable	



Inherited By IDMSourceOperation
IDMTargetOperation

Header File idmoper.hpp

Members				
Member	Page	Member	Page	
Constructor	128	setContainerRefreshOff	121	
addItem	123	setContainerRefreshOn	122	
containerObject	121	setDebugSupport	122	
copy	130	setDragInfo	129	
debugSupport	122	setDragResult	129	
drag	130	setOperation	125	
dragInfo	129	setPosition	126	
dragWasInterrupted	129	setSource	125	
item	123	setSourceWindowHandle	126	
link	130	setTargetWindowHandle	126	
move	130	source	126	
numberOfItems	123	sourceWindow	127	
operation	124	sourceWindowHandle	127	
position	126	targetWindow	127	
removeItem	123	targetWindowHandle	127	
replaceItem	124	unknown	130	
setContainerObject	121	~IDMOOperation	128	

The IDMOOperation class represents a drag operation in a direct manipulation. The drag operation is commonly known as a source or target operation. Objects of this class provide general-purpose information about a drag operation.

Objects of this class have the following attributes in addition to those inherited from its base class:

- The type of drag operation (for example, copy, move, or link)
- The position of the pointing device within the source or target window
- The source of the operation (pointing device versus keyboard)
- The number of associated drag items, IDMIItems
- A pointer to the collection of IDMIItems

IDMOperation

- A collection of IDItem objects, one object for each item being manipulated during this drag operation

This class manages a drag item collection that associates drag items with this class.

This class is an abstract base class. Your programs should only deal with objects of either of the derived classes: IDMSourceOperation (p. 153) and IDMTargetOperation (p. 207).

PM The IDMOperation class is a wrapper for the DRAGINFO structure.

Public Functions

Container Support

Use these members to support a container control in a direct manipulation.

containerObject

Returns a pointer to the container object that the hot spot of the pointing device is over during target enter event processing or the container object where the drop occurred. It returns 0 if one of the following conditions occur:

- The target window is not a container.
- The hot spot of the pointing device is positioned over container white space.
- The drop occurred on container white space.

IContainerObject*	Win	PM	Motif
containerObject() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setContainerObject

Sets a pointer to the container object that the hot spot of the pointing device is over during target enter event processing or to the container object where the drop occurred.

IDMOperation&	Win	PM	Motif
setContainerObject(IContainerObject* object);	<i>Y</i>	<i>Y</i>	<i>N</i>

object A pointer to the container object.

setContainerRefreshOff

Sets the container refresh off for the given container by calling IContainerControl::setRefreshOff (Vol. III).

IDMOperation

When direct manipulation processing is finished or if it is canceled, container refresh is turned on again via `IDMOperation::setRefreshOn` (p. 122).

Note: The User Interface Class Library manages container refreshing by default.

<code>IDMOperation&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setContainerRefreshOff(IContainerControl* container);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

container A pointer to the container control.

setContainerRefreshOn

Sets the container refresh on by calling `IContainerControl::setRefreshOn` (Vol. III). for the container that was identified in the previous call to `IDMOperation::setContainerRefreshOff` (p. 121). The container is also refreshed via the member function, `IContainerControl::refresh` (Vol. III).

The container is also refreshed.

Note: The User Interface Class Library manages container refreshing by default.

<code>IDMOperation&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setContainerRefreshOn();</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

Diagnostics

Use these members to indicate the level of debug support for a direct manipulation.

debugSupport

Returns a value indicating if full debug support is enabled. When true, screen locking during a direct manipulation is disabled. The default setting for this flag is false.

<code>static Boolean</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>debugSupport();</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

setDebugSupport

Set this flag to true to enable full debug support. When set to true, this flag disables screen locking during a direct manipulation.

<code>static void</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setDebugSupport(Boolean debugSupport = true);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMOperation

debugSupport

A value that enables or disables full debug support.

Drag Items

Use these members to add, remove, and query the IDMIItem objects involved in a direct manipulation.

addItem Associates another drag item with the operation.

```
IDMOperation&
addItem( const IDMIItem::Handle& newItem );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

newItem A reference to a handle of a drag item to add to the operation's collection.

item Returns the drag item with a given position. If the position is not valid, 0 is returned.

Note: Positions are 1-based.

```
IDMIItem::Handle
item( unsigned position );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

position A value that represents the position of the drag item in the operation's collection.

Exceptions	
InvalidRequest	The drag item handle was not returned. The specified index is not within the valid range.

numberOfItems

Returns the number of items involved in a direct manipulation. This number is the same as the number of drag items added to the operation's collection.

```
unsigned
numberOfItems();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

removeItem Removes a drag item from the operation.

```
1 IDMOperation&
removeItem( unsigned position );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMOperation

position A value that represents the position of the drag item to remove from the operation's collection.

Use this version of the function to remove a drag item using a position.

Note: Positions are 1-based.

2	<code>IDMOperation& removeItem(const IDMItem::Handle& itemToRemove);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

itemToRemove
A reference to a handle of a drag item to remove from the operation's collection.

Use this version of the function to remove a drag item using a reference to the drag item's handle.

replaceItem Replaces a given drag item with another specified drag item.

Note: Positions are 1-based.

<code>IDMOperation& replaceItem(unsigned position, const IDMItem::Handle& replacement);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

position A value that represents the position of the drag item to replace in the operation's collection.

replacement
A reference to a handle of a replacement drag item.

Exceptions	
InvalidRequest	The drag item was not replaced. The specified index is not within the valid range.

Drag Operations

Use these members to set and query the default drag operation of the direct manipulation.

operation Returns the default drag operation for the direct manipulation. The possible default drag operations include the following:

- `IDMOperation::copy` (p. 130)
- `IDMOperation::drag` (p. 130)
- `IDMOperation::link` (p. 130)
- `IDMOperation::move` (p. 130)
- `IDMOperation::unknown` (p. 130)

IDMOperation

Application-defined drag operations are assigned a value greater than `IDMOperation::unknown`.

```
virtual unsigned long  
    operation() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setOperation Sets the default drag operation for the direct manipulation. Use one of the following operations to identify the default drag operation:

- `IDMOperation::copy` (p. 130)
- `IDMOperation::drag` (p. 130)
- `IDMOperation::link` (p. 130)
- `IDMOperation::move` (p. 130)
- `IDMOperation::unknown` (p. 130)

Assign a value greater than `IDMOperation::unknown` when assigning an application-defined drag operation. Also, you must derive a new target renderer from `IDMTargetRenderer` (p. 213) and override `IDMTargetRenderer::supportsOperation` (p. 215) to support the new operation.

Note: When one of the default operations is set to an operation other than `IDMOperation::drag`, that operation precludes the use of the augmentation key defined for the operation. For example, `IDMOperation::copy` allows the copy operation to function without the use of the copy augmentation key that is by default the Ctrl key.

```
virtual IDMOperation&  
    setOperation( unsigned long op );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

op A value that represents the drag operation.

Drag Source Device

Use these members to set and query the type of device that is used to initiate a direct manipulation.

setSource Sets the type of the device that is the source of the direct manipulation.

Note: Generally, you do not call this function. The operating system determines the source of a direct manipulation, and the User Interface Class Library uses this function to save that information.

```
virtual IDMOperation&  
    setSource( Source source );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMOperation

source A value that represents the source of a direct manipulation.

PM The source of a direct manipulation is always the pointing device, `IDM::pointingDevice`.

source Returns the type of the device that is the source of the direct manipulation.

<code>virtual Source</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>source() const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

PM The source of a direct manipulation is always the pointing device, `IDM::pointingDevice`.

Pointing Device Position

Use these members to set or query the pointing device's position.

position Returns the hot spot of the pointing device within the source or target window.

<code>virtual IPoint</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>position() const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

setPosition Sets the hot spot of the pointing device within the source or target window.

<code>virtual IDMOperation&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setPosition(IPoint position);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

position Points that represent the hot spot of the pointing device.

Source and Target Window Support

Use these members to set and query the source and target windows.

setSourceWindowHandle

Sets the source window handle for a direct manipulation.

<code>virtual IDMOperation&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setSourceWindowHandle(IWindowHandle window);</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

window The handle of the source window.

setTargetWindowHandle

Sets the target window handle for a direct manipulation.

IDMOperation

```
virtual IDMOperation&
    setTargetWindowHandle( IWindowHandle window );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

window The handle of the target window.

sourceWindow

Returns a pointer to the source window for a direct manipulation. Returns 0 if the window is not valid.

```
virtual IWindow*
    sourceWindow() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sourceWindowHandle

Returns the handle of the source window for a direct manipulation. Returns 0 if the window is not valid.

```
virtual IWindowHandle
    sourceWindowHandle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

targetWindow

Returns a pointer to the target window for a direct manipulation. Returns 0 if the window is not valid. For example, if the source window attempts to access the target window before the target window is known, the window is not valid.

```
virtual IWindow*
    targetWindow() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

targetWindowHandle

Returns the handle of the target window for a direct manipulation. Returns 0 if the window is not valid. For example, if the source window attempts to access the target window before the target window is known, the window is not valid.

```
virtual IWindowHandle
    targetWindowHandle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IRefCounted		
addRef	removeRef	useCount

IDMOperation

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class. These members are used by derived classes.

IDMOperation

1

IDMOperation(IDMTargetEnterEvent& event);

Win
Y

PM
Y

Motif
N

event A reference to a target enter event.

Use this function to construct an IDMOperation object by providing a reference to a target enter event.

2

IDMOperation(IDMSourceBeginEvent& event);

Win
Y

PM
Y

Motif
N

event A reference to a source begin event.

Use this function to construct an IDMOperation object by providing a reference to the source begin event.

~IDMOperation

This destructor deletes the drag item collection that manages associated drag items for this class.

virtual

~IDMOperation();

Win
Y

PM
Y

Motif
N

IDMOperation

Operation Services

Use these members for various utility services.

dragInfo Returns the pointer to the drag information structure.

<code>_DRAGINFO* dragInfo() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

PM The return value is a pointer to the DRAGINFO structure.

dragWasInterrupted

Returns true if the direct manipulation is interrupted. Interrupts can occur when you do one of the following:

- Cancel the direct manipulation by pressing the Esc key
- Request help while dragging over an object
- Cancel the direct manipulation if the drop is rejected

<code>Boolean dragWasInterrupted() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDragInfo Sets the pointer to the drag information structure.

<code>IDMOperation& setDragInfo(_DRAGINFO* dragInfo);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

dragInfo A pointer to the drag information structure.

PM The parameter, *dragInfo*, is a pointer to the DRAGINFO structure.

setDragResult

Sets the overall result of a direct manipulation. A *result* of true indicates a successful direct manipulation and false indicates one of the following conditions:

- The direct manipulation is canceled by pressing the Esc key.
- Help is requested while dragging over an object.
- The direct manipulation is canceled if the drop is rejected.

<code>IDMOperation& setDragResult(Boolean result);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

result Boolean value that represents the direct manipulation result.

IDMOperation

Public Data

Drag Operations

Use these members to set and query the default drag operation of the direct manipulation.

copy	Copy operation (the copy augmentation key is pressed).			
	static const unsigned long copy;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
drag	Drag operation (no augmentation keys are pressed). This is the default operation.			
	static const unsigned long drag;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
link	Link operation (the link augmentation keys are pressed).			
	static const unsigned long link;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
move	Move operation (the move augmentation key is pressed).			
	static const unsigned long move;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
unknown	Unknown operation (application-defined operation codes should be assigned values greater than this value).			
	static const unsigned long unknown;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMRenderer

Derivation IBase
IVBase
IDMRenderer

Inherited By IDMSourceRenderer
IDMTargetRenderer

Header File idmrendr.hpp

Members	Member	Page	Member	Page
	Constructor	132	setSupportedRMFs	134
	defaultSourceRenderer	133	setSupportedTypes	134
	defaultTargetRenderer	133	supportedRMFs	134
	setDefaultSourceRenderer	133	supportedTypes	134
	setDefaultTargetRenderer	133	~IDMRenderer	132

The IDMRenderer class is the base class for the direct manipulation source and target renderer classes. Objects of this class render drag items during a direct manipulation. *Rendering* is the transfer of the data represented by the drag item from the source to the target.

The IDMRenderer class defines the common protocol for both source and target renderers. The derived classes, IDMSourceRenderer (p. 167) and IDMTargetRenderer (p. 213), define the specific protocol for source renderers and target renderers, respectively. The source and target renderers implement the protocol defined by the rendering mechanisms and formats (RMFs) to render the data represented by the drag items.

A generic renderer object has the following attributes:

- A set of supported drag item types with which this renderer can work. IDM::Type (p. 31) lists the default drag item types. You can also define your own types.
- A set of supported rendering mechanisms and formats (RMFs). These RMFs describe the various means this renderer can use to transfer the data represented by the drag item. IDM::RM (p. 33) and IDM::RF (p. 32) list the default rendering mechanisms and rendering formats, respectively. You can also define your own RMFs.

IDMRenderer

You can perform two categories of operations on objects of this class:

- Queries to determine whether the renderer can render a given drag item
- Requests to perform one of the basic rendering processes

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMRenderer

1	<code>IDMRenderer(const char* rmfs, const char* types);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
	<i>rmfs</i>	A pointer to the RMFs string.		
	<i>types</i>	A pointer to the drag item types string.		

Use this constructor to create a special-purpose renderer that renders drag items of a specific type using a specific RMF. You can construct a renderer by providing rendering mechanisms and formats (RMFs) and drag item types that indicate the drag items that can be rendered by this renderer.

2	<code>IDMRenderer();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	-----------------------------	-------------------------------	------------------------------	---------------------------------

Use this constructor, the default constructor, to create a general-purpose renderer that supports all drag item types using any library-supported RMF.

~IDMRenderer

<code>virtual ~IDMRenderer();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

Default Renderers

Use these members to set and query the default source and target renderers. The default User Interface Class Library source and target renderers are added automatically to newly created source and target handlers.

IDMRenderer

defaultSourceRenderer

Returns the default source renderer, which is an IDMSourceRenderer (p. 167) object, unless the default has been set to a derived IDMSourceRenderer object.

<pre>static IDMSourceRenderer& defaultSourceRenderer();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultTargetRenderer

Returns the default target renderer, which is an IDMTargetRenderer (p. 213) object, unless the default has been set to a derived IDMTargetRenderer object.

<pre>static IDMTargetRenderer& defaultTargetRenderer();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultSourceRenderer

Sets the default source renderer. Use this function to set the default source renderer prior to enabling the source for direct manipulation. Enable the source for direct manipulation with either IDMHandler::enableDragFrom (p. 52) or IDMHandler::enableDragDropFor (p. 50).

<pre>static void setDefaultSourceRenderer(IDMSourceRenderer& sourceRenderer);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

sourceRenderer

A reference to the source renderer object.

setDefaultTargetRenderer

Sets the default target renderer. Use this function to set the default target renderer prior to enabling the target for direct manipulation. Enable the target for direct manipulation with either IDMHandler::enableDropOn (p. 53) or IDMHandler::enableDragDropFor (p. 50).

<pre>static void setDefaultTargetRenderer(IDMTargetRenderer& targetRenderer);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

targetRenderer

A reference to the target renderer object.

IDMRenderer

Supported Types and RMFs

Use the following members to set and query the drag item types and rendering mechanisms and formats (RMFs) supported by renderers.

setSupportedRMFs

Sets the RMFs supported by the renderer.

```
virtual IDMRenderer&
    setSupportedRMFs( const char* rmfs );
```

rmfs A pointer to the RMFs string.

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setSupportedTypes

Sets the drag item types supported by the renderer.

```
virtual IDMRenderer&
    setSupportedTypes( const char* types );
```

types A pointer to the drag item types string.

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

supportedRMFs

Returns the RMFs supported by the renderer.

```
virtual IString
    supportedRMFs() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

supportedTypes

Returns the drag item types supported by the renderer.

```
virtual IString
    supportedTypes() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IDMRenderer

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	136	offset	137
	container	137	position	137
	containerId	137	source	137
	object	137	~IDMSourceBeginEvent	137

The IDMSourceBeginEvent class represents the event that is generated by the start of a direct manipulation at the source. Objects of this class are created by the source handler and passed as parameters to functions that participate in the handling of this event.

PM Objects of this class wrapper the WM_BEGINDRAG message and the container notification CN_INITDRAG.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceBeginEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMSourceBeginEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

event A reference to the event object.

IDMSourceBeginEvent

~IDMSourceBeginEvent

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IDMSourceBeginEvent();	<i>Y</i>	<i>Y</i>	<i>N</i>

Container Support

Use these members to query information about a container control that is the source of a direct manipulation.

container Returns a pointer to the source container from which the source begin event occurred.

virtual IContainerControl*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
container() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

containerId Returns the source container's identifier (ID).

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
containerId() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

object Returns a pointer to the object over which the drag started. If the drag started over the container's white space or if the window is not a container, 0 is returned.

Note: The white space of a container is an area that no container objects occupy.

virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
object() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

offset Returns the position of the hot spot of the pointing device relative to the source container object's origin.

virtual ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
offset() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to query event information for this class.

position Returns the position of the hot spot of the pointing device.

virtual IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
position() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

source Returns the source of the direct manipulation.

IDMSourceBeginEvent

```
virtual Source  
source() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The source of the direct manipulation is always IDM::pointingDevice.

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMSourceDiscardEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDMEvent
        IDMSourceDiscardEvent
  
```

Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	139	sourceOperation	140
	setSourceOperation	140	whoDiscards	141
	setWhoDiscards	140	~IDMSourceDiscardEvent	140

The IDMSourceDiscardEvent class represents objects that have been dropped on a shredder object. Objects of this class are constructed by the source handler, IDMSourceHandler (p. 145).

Objects of this class return an indicator to the shredder object, which identifies what has the responsibility for the deletion of the items or which indicates that the discard operation is to be aborted.

PM Objects of this class wrapper the DM_DISCARDOBJECT message.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceDiscardEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMSourceDiscardEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

event A reference to the event object.

IDMSourceDiscardEvent

~IDMSourceDiscardEvent

```
virtual  
~IDMSourceDiscardEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

Event Information

Use these members to set and query event information for this class.

setSourceOperation

Sets the handle of the drag source operation for this event.

```
IDMSourceDiscardEvent&  
setSourceOperation(  
    const IDMSourceOperation::Handle& operation);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

operation A reference to the handle of the source operation object.

setWhoDiscards

Sets the discard indicator, which identifies the following:

- The source discards the item. This is the default.

This is usually accomplished by calling `IDMItem::sourceDiscard` (p. 85) for the drag item.

- The target discards the item.

The target is usually the shredder object. Let it discard the object only if the object is a file; otherwise, it does not recognize how to discard your application objects.

- The discard is aborted.

```
IDMSourceDiscardEvent&  
setWhoDiscards( DiscardIndicator indicator );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

indicator A value that represents the discard indicator.

sourceOperation

Returns the handle of the drag source operation for this event.

```
IDMSourceOperation::Handle  
sourceOperation() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

IDMSourceDiscardEvent

whoDiscards Returns the discard indicator identifying what should delete the drag items.

```
virtual DiscardIndicator  
whoDiscards() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

Inherited Public Functions

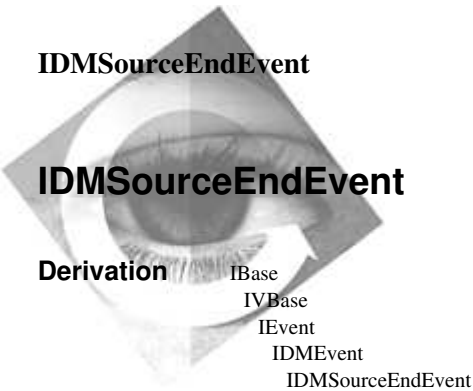
IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	142	wasTargetSuccessful	143
	dragItem	143	~IDMSourceEndEvent	142

The IDMSourceEndEvent class represents objects that the target sends to the source when the target renderer completes the rendering of a drag item. Objects of this class are constructed by the source handler, IDMSourceHandler (p. 145). An object of this class is constructed for each drag item when a target renderer completes rendering.

PM Objects of this class wrapper the DM_ENDCONVERSATION message.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceEndEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMSourceEndEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>
<i>event</i>	A reference to the event object.		

~IDMSourceEndEvent

IDMSourceEndEvent

```
virtual                                Win PM Motif  
    ~IDMSourceEndEvent();              Y   Y   N
```

Rendering

Use these members to query information about the rendering process.

dragItem Returns the handle of drag item which the target has completed rendering.

```
virtual IDMIItem::Handle                Win PM Motif  
    dragItem() const;                  Y   Y   N
```

wasTargetSuccessful

Returns true if the target successfully completed rendering the drag item.

```
virtual Boolean                          Win PM Motif  
    wasTargetSuccessful() const;       Y   Y   N
```

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IDMSourceEndEvent

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMSourceHandler

Derivation

```

IBase
IVBase
IHandler
IDMHandler
IDMSourceHandler
  
```

Inherited By None.

Header File idmsrch.hpp

Members	Member	Page	Member	Page
	Constructor	145	replaceRenderer	148
	addRenderer	147	setRenderer	148
	allocateOperation	151	sourceBegin	149
	dispatchHandlerEvent	149	sourceDiscard	149
	findRendererFor	151	sourceEnd	150
	findRenderersFor	151	sourcePrepare	150
	numberOfRenderers	147	sourcePrint	150
	removeRenderer	147	sourceRender	150
	renderer	147	~IDMSourceHandler	146

The IDMSourceHandler class processes events occurring at the source of a direct manipulation. You must add an object of this class to a window so it can support source direct manipulation. Objects of this class do not allow objects to be dropped on a window. See IDMTargetHandler (p. 195) for information about dropping objects on a window.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceHandler

	IDMSourceHandler(IContainerControl* containerControl);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

IDMSourceHandler

containerControl

A pointer to the container control object.

Creates an object of this class using a pointer to a container control object. The constructed handler is automatically attached to the specified container.

2	IDMSourceHandler(IWindow* window);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

window A pointer to the window object.

Creates an object of this class using a pointer to an IWindow-based object that represents a control that is not supported by the default User Interface Class Library support for direct manipulation. These controls can be system-defined controls or custom controls. The constructed handler is automatically attached to the specified window.

3	IDMSourceHandler(IEntryField* entryField);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

entryField A pointer to the entry field object.

Creates an object of this class using a pointer to an entry field object. The constructed handler is automatically attached to the specified entry field.

4	IDMSourceHandler(IMultiLineEdit* multiLineEdit);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

multiLineEdit

A pointer to the multiline edit object.

Creates an object of this class using a pointer to a multiline edit (MLE) object. The constructed handler is automatically attached to the specified MLE.

5	IDMSourceHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This is the default constructor.

~IDMSourceHandler

virtual	~IDMSourceHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

IDMSourceHandler

Renderer Processing

These members override the rendering functions defined in the base IDMHandler (p. 48) class. Use them to access the IDMSourceRenderer (p. 167) objects involved in a direct manipulation. Objects of this class render the drag item objects that are compatible with the rendering mechanisms and formats that the User Interface Class Library supports.

addRenderer Adds the specified renderer to the set of candidates maintained by this handler and uses it to render objects dragged from a source window where this handler is attached. The renderer is added to the end of the renderer collection.

Note: Renderers are maintained by position, which are 1-based.

<code>virtual IDMSourceHandler& addRenderer(const IDMSourceRenderer& newRenderer);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

newRenderer

A reference to the renderer object.

numberOfRenderers

Returns the number of renderers associated with the source handler.

<code>virtual unsigned numberOfRenderers();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeRenderer

Removes a specified renderer from this handler.

Note: Renderers are maintained by position, which are 1-based.

<code>virtual IDMSourceHandler& removeRenderer(const IDMSourceRenderer& rendererToRemove);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

rendererToRemove

A reference to the source renderer object to remove from the renderer collection.

renderer

Returns the renderer with the specified position.

Note: Renderers are maintained by position, which are 1-based.

<code>virtual IDMSourceRenderer* renderer(unsigned position);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMSourceHandler

position A value that represents the location of the source renderer.

replaceRenderer

Replaces the given renderer with another specified renderer.

Note: Renderers are maintained by position, which are 1-based.

```
virtual IDMSourceHandler&
    replaceRenderer( unsigned position,
                    const IDMSourceRenderer& replacement );
```

Win PM Motif
Y Y N

position A value that represents the location of the source renderer to replace in the renderer collection.

replacement
A reference to the replacement renderer object.

setRenderer Sets the source renderer for this handler. Removes any pre-existing source renderers.

```
virtual IDMSourceHandler&
    setRenderer( const IDMSourceRenderer& newRenderer );
```

Win PM Motif
Y Y N

newRenderer
A reference to the source renderer object.

Inherited Public Functions

IDMHandler		
defaultDragHandler	defaultTargetHandler	enableDropOn
defaultDropTargetHandler	enableDragDropFor	setDefaultSourceHandler
defaultSourceHandler	enableDragFrom	setDefaultTargetHandler

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IDMSourceHandler

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

These members dispatch the various events for a direct manipulation source.

dispatchHandlerEvent

Overrides the default handler's dispatcher. This function processes the direct manipulation events for the source.

```
virtual Boolean
    dispatchHandlerEvent( IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the event object.

Event Processing

These members are dispatched in response to specific direct manipulation events for the source.

sourceBegin Called to process the start of a direct manipulation in response to the system's begin drag event. Initial processing invokes `allocateOperation`. Next, a pointer to the drag item provider is obtained by calling `IWindow::itemProvider` (Vol. II). Afterwards, `IDMItemProvider::provideSourceItems` (p. 105) is invoked to retrieve the items to be dragged. `findRenderersFor` is called to find appropriate source renderers for the items that were provided. If there are items to be dragged, `IDMSourceOperation::begin` (p. 156) is called to start the direct manipulation.

```
virtual Boolean
    sourceBegin( IDMSourceBeginEvent& event,
                Boolean container = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source begin event object.
container A Boolean value that identifies the container as the source of the source begin event.

sourceDiscard

Called to indicate that a drag item or items have been dropped on a shredder object. The source response determines what deletes the items or if the discard operation is

IDMSourceHandler

cancelled. The default implementation dispatches IDMSourceRenderer::sourceDiscard (p. 169).

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceDiscard(IDMSourceDiscardEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source discard event object.

sourceEnd Called to indicate that a given item has been processed by the target. The default implementation for this class dispatches IDMSourceRenderer::sourceEnd (p. 169) on the corresponding drag item object.

After all source end events are received, the IDMSourceOperation (p. 153) object is deleted.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceEnd(IDMSourceEndEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source end event object.

sourcePrepare

Called to indicate to the source that rendering is ready to start. Requests for rendering preparation are made within the source item constructors. The default implementation in this class dispatches IDMSourceRenderer::sourcePrepare (p. 170) on the corresponding drag item object.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourcePrepare(IDMSourcePrepareEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source prepare event object.

sourcePrint Called to indicate that a drag item or items have been dropped on a printer object. The source response determines who prints the items or if the operation is cancelled. The default implementation dispatches IDMSourceRenderer::sourcePrint.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourcePrint(IDMSourcePrintEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source print event object.

sourceRender

IDMSourceHandler

Called to start the rendering of a drag item by the source renderer. The default implementation for this class dispatches IDMSourceRenderer::sourceRender (p. 170) on the drag item object.

virtual Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceRender(IDMSourceRenderEvent& event);		<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source render event object.

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

allocateOperation

Creates a new IDMSourceOperation (p. 153) object.

This function is called when the source handler is handling the processing of a new direct manipulation. Derived classes should override this function and return a class derived from IDMSourceOperation.

This function returns a handle to the source operation object.

virtual IDMSourceOperation::Handle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
allocateOperation(IDMSourceBeginEvent& event,		<i>Y</i>	<i>Y</i>	<i>N</i>
DragImageStyle dragItemStyle) const;				

event A reference to the source begin event object.

dragImageStyle

A value that identifies the drag image's style.

findRendererFor

Finds the appropriate source renderer for the drag item. Returns the position of the renderer or 0 if a renderer cannot be found.

virtual unsigned		<u>Win</u>	<u>PM</u>	<u>Motif</u>
findRendererFor(const IDItem::Handle& item);		<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to the handle of the drag item object.

findRenderersFor

Finds the appropriate source renderers for the specified source operation.

IDMSourceHandler

```
virtual Boolean  
    findRenderersFor(  
        const IDMSourceOperation::Handle& sourceOperation);
```

sourceOperation
A reference to the handle of the source operation object.

Inherited Protected Functions

IDMHandler		
addRenderer	numberOfRenderers	renderer
isContainerControl	removeRenderer	replaceRenderer

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMSourceOperation

Derivation IBase
 IVBase
 IRefCounted
 IDMOperation
 IDMSourceOperation

Inherited By None.

Header File idmsrcop.hpp

Members	Member	Page	Member	Page
	Constructor	153	setPointerOffset	154
	begin	156	setSourceOperation	157
	imageStyle	154	setStackingPercentage	155
	operation	156	sourceOperation	157
	operationFrom	157	stackingPercentage	155
	pointerOffset	154	~IDMSourceOperation	154
	setImageStyle	155		

The IDMSourceOperation class represents a source operation in a direct manipulation. Objects of this class provide information about source operations. You can access this information from a drag item using `IDMItem::sourceOperation` (p. 96).

Objects of this class are not intended to be created by application programmers. The source event handler class, `IDMSourceHandler` (p. 145) creates them.

You can access an object of this class using `IDMSourceOperation::Handle`, which reference-counts the object.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceOperation

Constructs objects of this class.

IDMSourceOperation

```
IDMSourceOperation(
    IDMSourceBeginEvent& event,
    DragImageStyle dragStyle = IDM::systemImages );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to a source begin event.
dragStyle The drag image style. The default is IDM::systemImages.

~IDMSourceOperation

Destructs objects of this class.

Note: If a container is the source of the direct manipulation, the container refresh is enabled and the container is refreshed.

```
virtual
~IDMSourceOperation();
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Container Support

Use these members to set and query container-specific attributes.

pointerOffset Returns the position of the hot spot of the pointing device relative to the source container object's origin. If the source is not a container, ISize(0, 0) is returned.

```
virtual ISize
pointerOffset() const;
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setPointerOffset

Sets the position of the hot spot of the pointing device relative to the source container object's origin.

```
virtual IDMSourceOperation&
setPointerOffset( const ISize& offset );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

offset A reference to the position of the hot spot of the pointing device relative to the source container's origin.

Drag Image Support

Use these functions to set and query the drag image style and the stacking percentage that determines the placement of stacked drag images.

imageStyle Returns the drag image style for the source operation.

IDMSourceOperation

```
virtual DragImageStyle  
    imageStyle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

setImageStyle

Sets the drag image style for the source operation.

```
virtual IDMSourceOperation&  
    setImageStyle(  
        DragImageStyle dragImageStyle = IDM::systemImages);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

dragImageStyle

The drag image style. The default is IDM::systemImages.

setStackingPercentage

Sets the stacking percentage that is used to calculate the placement of the next "stacked" image when the drag image style, IDM::stack3AndFade or IDM::allStacked, is specified.

The stacking percentage determines the placement of the next stacked image based upon the origin of the current image. For example, if you specify IPair(25, 25) as the stacking percentage, the origin of the next stacked image is positioned at a point that is located 25 percent of the current images' width and height plus the origin of the current image. For example, if the origin of the current image is (0,0) and its size is (32, 32), the origin of the next stacked image is (8, 8).

If you specify a negative value for either of the values that comprise the stacking percentage, you can vary the stacking direction.

Note: The origin of the initial image is relative to the hot spot of the pointing device. It is determined by the pointer offset value returned by IDMImage::pointerOffset (p. 65).

```
virtual IDMSourceOperation&  
    setStackingPercentage( const IPair& stackingPercentage );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

stackingPercentage

A reference to the stacking percentage.

stackingPercentage

Returns the stacking percentage that is used to calculate the placement of the next "stacked" image when the drag image style, IDM::stack3AndFade or IDM::allStacked, is specified.

IDMSourceOperation

IPair	<u>Win</u>	<u>PM</u>	<u>Motif</u>
stackingPercentage() const;	<i>I</i>	<i>Y</i>	<i>N</i>

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

begin Initiates the direct manipulation.

Note: Generally, you do not call this function. This function is called by the IDMSourceHandler (p. 145) object during the processing of a source begin event.

IDMSourceOperation&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
begin();	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The drag operation was not started. The drag information could not be allocated. The system resources may be exhausted.
IAccessError	The drag operation was not started. Allocation of a drag item failed.
IAccessError	The drag operation was not started. A drag item may be invalid.

Operation Services

Use these members for various operational services.

operation Returns the default drag operation for the direct manipulation. Use this function to determine the actual operation that occurred at the target after the drop has occurred.

The possible default drag operations include the following:

- IDMOperation::copy (p. 130)
- IDMOperation::drag (p. 130)
- IDMOperation::link (p. 130)
- IDMOperation::move (p. 130)
- IDMOperation::unknown (p. 130)

Application-defined drag operations are assigned a value greater than IDMOperation::unknown.

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operation() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

IDMSourceOperation

operationFrom

Retrieves a handle to the source operation from the drag information structure.

```
static Handle  
    operationFrom( _DRAGINFO* dragInformation );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

dragInformation

A pointer to the drag information structure.



The parameter, *dragInformation*, is a pointer to the DRAGINFO structure.

Exceptions	
IAccessError	The source operation handle was not retrieved. The drag information is not accessible.

setSourceOperation

Sets an IDMSourceOperation handle during the processing of the source begin event.

```
static void  
    setSourceOperation(  
        IDMSourceOperation::Handle srcOperation);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

srcOperation

A handle to a source operation object.

sourceOperation

Returns the IDMSourceOperation handle created during the processing of the source begin event.

```
static IDMSourceOperation::Handle  
    sourceOperation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

IDMOperation		
addItem	removeItem	setPosition
containerObject	replaceItem	setSource
debugSupport	setContainerObject	setSourceWindowHandle
item	setContainerRefreshOff	setTargetWindowHandle
numberOfItems	setContainerRefreshOn	source
operation	setDebugSupport	sourceWindow

IDMSourceOperation

IDMOperation		
position	setOperation	sourceWindowHandle

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMOperation		
dragInfo	dragWasInterrupted	setDragInfo

Inherited Public Data

IDMOperation		
copy	drag	link

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Type Definitions

Handle `typedef IReference < IDMSourceOperation > Handle;`

IDMSourceOperation::Handle provides access to the IDMSourceOperation objects associated with a direct manipulation. This handle manages the references to the

IDMSourceOperation

IDMSourceOperation object and ensures that this object is not destructed until the direct manipulation is completed.

Use Handle to reference a source operation handle within this class and IDMSourceOperation::Handle externally.

This handle provides a pointer operator, which enables the handle to be treated as a pointer to an IDMSourceOperation object.

IDMSourcePrepareEvent

IDMSourcePrepareEvent

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IEvent
    IEvent --> IDMEvent
    IDMEvent --> IDMSourceRenderEvent
    IDMSourceRenderEvent --> IDMSourcePrepareEvent
```

Inherited By None.

Header File idmevent.hpp

Members

Member	Page	Member	Page
Constructor	161	setNoSourceRendering	162
alternateWindow	161	setTargetCanRetry	162
alternateWindowHandle	161	targetCanRetry	162
noSourceRendering	162	~IDMSourcePrepareEvent	161
setAlternateWindowHandle	161		

The IDMSourcePrepareEvent class represents objects that the target sends to the source when rendering preparation is required for a drag item prior to source rendering for the item. Objects of this class are constructed by the source handler, IDMSourceHandler (p. 145). An object of this class is constructed for each drag item when source preparation is required.

Objects of this class provide all of the attributes inherited from IDMSourceRenderEvent (p. 174). In addition, functions are provided to return the following information to the target:

- An alternate source window with which the target can communicate during rendering
- Two result flags indicating whether the target does the following:
 - Performs the rendering
 - Retries the rendering with a different rendering mechanism and format



Objects of this class wrapper the DM_RENDERPREPARE message.

Public Functions

IDMSourcePrepareEvent

Alternate Window

Use these members to set and query the alternate source window. For example, the alternate source window can be used to implement source rendering on a secondary thread.

alternateWindow

Returns a pointer to the alternate source window object.

<code>virtual IWindow*</code> <code>alternateWindow() const;</code>	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

alternateWindowHandle

Returns the handle of the alternate source window object.

<code>virtual IWindowHandle</code> <code>alternateWindowHandle() const;</code>	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	-------------------------------	------------------------------	---------------------------------

setAlternateWindowHandle

Sets the handle of the alternate source window object.

<code>IDMSourcePrepareEvent&</code> <code>setAlternateWindowHandle(const IWindowHandle& window);</code>	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

window A reference to the handle of the alternate source window object.

Constructors

You can construct and destruct objects of this class.

IDMSourcePrepareEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

<code>IDMSourcePrepareEvent(const IEvent& event);</code>	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

event A reference to the event object.

~IDMSourcePrepareEvent

<code>virtual</code> <code>~IDMSourcePrepareEvent();</code>	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

IDMSourcePrepareEvent

Rendering

Use these members to provide the render preparation support. The results of the preparation are returned to the target.

noSourceRendering

Returns true if only the target can perform the rendering. The default is false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noSourceRendering() const;	<i>I</i>	<i>Y</i>	<i>I</i>

setNoSourceRendering

Sets the no-source-rendering flag that indicates that only the target can perform the rendering.

IDMSourcePrepareEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setNoSourceRendering(Boolean flag);	<i>I</i>	<i>Y</i>	<i>I</i>

flag A Boolean value that indicates if only the target can perform the rendering.

setTargetCanRetry

Sets the retry flag that indicates if the target can retry a failed rendering with a different rendering mechanism and format.

IDMSourcePrepareEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setTargetCanRetry(Boolean flag);	<i>I</i>	<i>Y</i>	<i>I</i>

flag A Boolean value that indicates if the target can retry a failed rendering with a different rendering mechanism and format.

targetCanRetry

Returns true if the target can retry a failed rendering with a different rendering mechanism and format. The default is false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetCanRetry() const;	<i>I</i>	<i>Y</i>	<i>I</i>

Inherited Public Functions

IDMSourceRenderEvent		
alternateWindow	dragItem	setRetry
alternateWindowHandle	selectedFormat	setTargetInfo
canRetry	selectedMechanism	targetInfo
completion	setCompletion	targetName

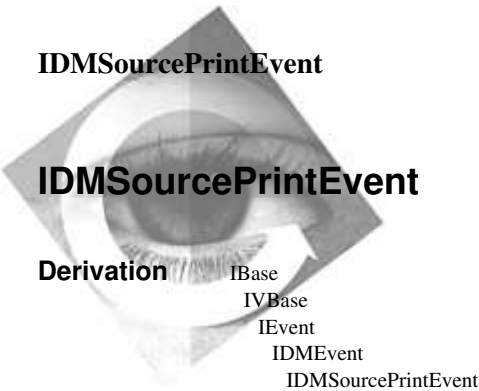
IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	164	sourceOperation	166
	printDestination	165	whoPrints	166
	setSourceOperation	165	~IDMSourcePrintEvent	165
	setWhoPrints	165		

The IDMSourcePrintEvent class represents objects that have been dropped on a print object. Objects of this class are constructed by the source handler, IDMSourceHandler (p. 145).

Objects of this class return an indicator to the print object, which identifies what has the responsibility for printing the items or which indicates that the print operation is to be aborted.

PM Objects of this class wrapper the DM_PRINTOBJECT message.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourcePrintEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMSourcePrintEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>I</i>
<i>event</i>	A reference to the event object.		

IDMSourcePrintEvent

Exceptions	
InvalidRequest	The object for the source print event was not created. The drag item may be invalid.

~IDMSourcePrintEvent

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IDMSourcePrintEvent();	<i>I</i>	<i>Y</i>	<i>I</i>

Print Support

Use these members to set and query printing attributes.

printDestination

Returns the pointer to the print destination structure.

_PRINTDEST*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
printDestination() const;	<i>I</i>	<i>Y</i>	<i>I</i>

PM The return value is a pointer to the PRINTDEST structure.

setSourceOperation

Sets the handle of the drag source operation for this event.

IDMSourcePrintEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setSourceOperation(<i>I</i>	<i>Y</i>	<i>I</i>
const IDMSourceOperation::Handle& operation);			

operation A reference to the handle of the source operation object.

setWhoPrints

Sets the print indicator, which identifies the following:

- The source prints the item. This is the default.
- The target prints the item.
- The print operation is aborted.

IDMSourcePrintEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setWhoPrints(PrintIndicator indicator);	<i>I</i>	<i>Y</i>	<i>I</i>

indicator A value that represents the print indicator.

IDMSourcePrintEvent

sourceOperation

Returns the handle to the drag source operation for this event.

IDMSourceOperation::Handle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceOperation() const;	<i>I</i>	<i>Y</i>	<i>I</i>

whoPrints

Returns the print indicator identifying what should print the drag items.

virtual PrintIndicator	<u>Win</u>	<u>PM</u>	<u>Motif</u>
whoPrints() const;	<i>I</i>	<i>Y</i>	<i>I</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter 2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMSourceRenderer

Derivation IBase
 IVBase
 IDMRenderer
 IDMSourceRenderer

Inherited By None.

Header File idmsrcrn.hpp

Members	Member	Page	Member	Page
	Constructor	167	informTargetOfCompletion	169
	canRender	169	sourceDiscard	169
	doDiscard	171	sourceEnd	169
	doPrint	171	sourcePrepare	170
	doRender	172	sourcePrint	170
	doRenderEnd	172	sourceRender	170
	doRenderPrepare	172	~IDMSourceRenderer	168

The IDMSourceRenderer class provides source rendering support for a direct manipulation. Objects of this class are registered with the source handler via IDMRenderer::setDefaultSourceRenderer (p. 133), IDMSourceHandler::addRenderer (p. 147), or IDMSourceHandler::setRenderer (p. 148).

The source handler selects a best-match source renderer by calling IDMSourceHandler::findRendererFor (p. 151) for a given drag item and invokes the rendering functions of this renderer when source rendering events occur.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceRenderer

1	IDMSourceRenderer(const char* rmfs, const char* type = IDM::any);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td>Y</td> <td>Y</td> <td>N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						
<i>rmfs</i>	A pointer to the RMFs string.							

IDMSourceRenderer

type A pointer to the drag item types string. The default is any drag item type, IDM::any.

Use this function to construct objects of this class by providing rendering mechanisms and formats (RMFs) and drag item types that indicate the drag items that can be rendered by this source renderer.

2	IDMSourceRenderer();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This is the default constructor.

When this constructor is used, the renderer supports any drag item type and all of the default source RMFs supported by the User Interface Class Library:

```
(IDM::rmLibrary)x(IDM::rfProcess,IDM::rfText,IDM::rfSharedMem),  
(IDM::rmDiscard,IDM::rmPrint)x(IDM::unKnown), <IDM::rmFile,IDM::rfText>
```

~IDMSourceRenderer

<i>virtual</i>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IDMSourceRenderer();		<i>Y</i>	<i>Y</i>	<i>N</i>

Source and Target Rendering

Use these members to support the implementation of source rendering. Members that you can use to support target rendering are identified on an individual basis.

Source rendering is rendering that involves both the source and target after the drop has occurred. The data for rendering is not available when the drop occurs, so the target requests that the source make the data available. The request for data is issued for each drag item using IDMSourceRenderEvent (p. 174) and possibly IDMSourcePrepareEvent (p. 160). The source can notify the target when it has finished rendering each drag item using IDMTargetEndEvent (p. 183). The target informs the source when it has finished rendering each drag item via IDMSourceEndEvent (p. 142).

Note: The default source rendering support in the User Interface Class Library does not use IDMTargetEndEvent.

Target rendering, the optimal form of rendering, involves mainly the target after the drop has occurred. The data for rendering is available when the drop occurs. The target informs the source when it has finished rendering each drag item via IDMSourceEndEvent (p. 142).

IDMSourceRenderer

canRender Returns true if the source renderer can render drag items with the specified attributes. This function is called for each drag item during source begin event processing.

Note: This function can be used for either source or target rendering.

```
virtual Boolean  
    canRender( const IString& types );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

types A reference to the drag item types string.

informTargetOfCompletion

Informs the target when source rendering has completed on the drag item. It is not used by any of the default renderers supported by the User Interface Class Library.

Note: This function is used for source rendering only.

```
virtual IDMSourceRenderer&  
    informTargetOfCompletion( IDMSourceRenderEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source render event object.

Exceptions	
InvalidRequest	The target could not be informed that source rendering completed. The drag transfer information is invalid.
IAccessError	The target could not be informed that source rendering completed. Posting the target end event to the target failed.

sourceDiscard

Called by IDMSourceHandler::sourceDiscard (p. 149) when the source has responsibility for discarding (deleting) the drag item.

```
virtual IDMSourceRenderer&  
    sourceDiscard( IDMSourceDiscardEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source discard event object.

sourceEnd

Called when the target informs the source that it has finished processing the data associated with the dropped item.

Note: This function can be used for either source or target rendering.

```
virtual IDMSourceRenderer&  
    sourceEnd( IDMSourceEndEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMSourceRenderer

event A reference to the source end event object.

sourcePrepare

Called when a drag item indicates it requires preparation, which can be any activity the source needs to perform before rendering the data.

For example, the drag item may require the source to create a secondary thread when source rendering occurs. This allows the system to remain responsive to the users.

Note: This function is used for source rendering only.

```
virtual IDMSourceRenderer&
    sourcePrepare( IDMSourcePrepareEvent& event );
```

Win **PM** **Motif**
Y *Y* *N*

event A reference to the source prepare event object.

sourcePrint

Called by IDMSourceHandler::sourcePrint (p. 150) when the source has responsibility for printing the item.

```
virtual IDMSourceRenderer&
    sourcePrint( IDMSourcePrintEvent& event );
```

Win **PM** **Motif**
Y *Y* *N*

sourceRender

Called when the target renderer requests that a source renderer render a drag item.

Note: This function is used for source rendering only.

```
virtual IDMSourceRenderer&
    sourceRender( IDMSourceRenderEvent& event );
```

Win **PM** **Motif**
Y *Y* *N*

event A reference to the source render event object.

Exceptions	
InvalidRequest	The source renderer could not render the item. The rendering format is invalid.
InvalidRequest	The source renderer could not render the item. The rendering mechanism is invalid.

Inherited Public Functions

IDMRenderer		
defaultSourceRenderer	setDefaultSourceRenderer	setSupportedRMFs

IDMSourceRenderer

IDMRenderer		
defaultTargetRenderer	setDefaultTargetRenderer	setSupportedTypes

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Renderer Specifics

Use these members to process the different rendering operations that are specific to the supported rendering mechanisms and formats.

doDiscard Called when a source discard event occurs and this renderer must discard (delete) a drag item.

Note: The User Interface Class Library performs no default processing in this function.

```
virtual Boolean
doDiscard( IDMSourceDiscardEvent& event );
```

Win

PM

Motif

Y

Y

N

event A reference to the source discard event object.

PM The OS/2 Workplace Shell sends this event via the DM_DISCARDOBJECT message whenever a drop occurs on a shredder object.

doPrint Called when a source print event occurs and this renderer must print a drag item.

Note: The User Interface Class Library performs no default processing in this function.

```
virtual Boolean
doPrint( IDMSourcePrintEvent& event );
```

Win

PM

Motif

Y

Y

N

IDMSourceRenderer

event A reference to the source print event object.



The OS/2 Workplace Shell sends this event via the DM_PRINTOBJECT message whenever a drop occurs on a printer object.

doRender

Called when a source render event occurs and this renderer must use source rendering to render a drag item.

The User Interface Class Library uses this function to implement the support for the <IDM::rmLibrary,IDM::rfSharedMem> rendering mechanism and format.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
doRender(IDMSourceRenderEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source render event object.

doRenderEnd

Called when a source end event occurs and this renderer must end the render operation on a drag item.

Note: The User Interface Class Library performs no default processing in this function.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
doRenderEnd(IDMSourceEndEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source end event object.

doRenderPrepare

Called when a render prepare event occurs and this renderer must use source rendering to render a drag item.

The User Interface Class Library uses this function to implement the support for the <IDM::rmLibrary,IDM::rfSharedMem> rendering mechanism and format.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
doRenderPrepare(IDMSourcePrepareEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the source prepare event object.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IDMSourceRenderEvent

IDMSourceRenderEvent

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IEvent
    IEvent --> IDMEvent
    IDMEvent --> IDMSourceRenderEvent
```

Inherited By IDMSourcePrepareEvent

Header File idmevent.hpp

Members

Member	Page	Member	Page
Constructor	175	setCompletion	176
alternateWindow	175	setRetry	176
alternateWindowHandle	175	setTargetInfo	177
canRetry	176	targetInfo	177
completion	176	targetName	177
dragItem	176	targetWindow	177
selectedFormat	176	targetWindowHandle	177
selectedMechanism	176	~IDMSourceRenderEvent	175

The IDMSourceRenderEvent class represents objects that the target sends to the source when the target renderer requests source rendering of a drag item. Objects of this class are constructed by the source handler, IDMSourceHandler (p. 145). An object of this class is constructed for each drag item when a target renderer requests source rendering.

In addition to the standard IEvent attributes of event and window identifiers, objects of this class also have the following attributes:

- An associated drag item object
- An alternate window handle
- A target window handle
- A rendering mechanism and format (RMF) selected by the target
- A target name
- Target-defined information



Objects of this class wrapper the DM_RENDER message.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMSourceRenderEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMSourceRenderEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

event A reference to the event object.

Exceptions	
InvalidRequest	The object for the source render event was not created. The drag transfer information may be invalid.

~IDMSourceRenderEvent

```
virtual  
~IDMSourceRenderEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

Rendering

Use these members to set and query attributes related to source rendering.

alternateWindow

Returns a pointer to the alternate source window object. For example, the alternate window can be used to implement source rendering on a secondary thread.

```
virtual IWindow*  
alternateWindow() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

alternateWindowHandle

Returns the handle of the alternate source window object. For example, the alternate window can be used to implement source rendering on a secondary thread.

```
virtual IWindowHandle  
alternateWindowHandle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

IDMSourceRenderEvent

canRetry Returns true if the renderer processing the event can retry processing the event if a failure occurs. This flag is passed to the target renderer by IDMSourceRenderer::informTargetOfCompletion (p. 169). The default is false.

virtual Boolean canRetry() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--------------------------------------	------------------------	-----------------------	--------------------------

completion Returns the flag that indicates if the rendering completed. This flag is passed to the target renderer by IDMSourceRenderer::informTargetOfCompletion (p. 169). The default is IDM::renderOk.

virtual IDM::RenderCompletion completion() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

dragItem Returns the handle of the drag item that is being rendered.

virtual IDItem::Handle dragItem() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

selectedFormat Returns the selected rendering format.

virtual IString selectedFormat() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

selectedMechanism Returns the selected rendering mechanism.

virtual IString selectedMechanism() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

setCompletion Sets the completion flag that indicates if the rendering completed successfully.

virtual IDMSourceRenderEvent& setCompletion(IDM::RenderCompletion code);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

code A value that represents the rendering completion code.

setRetry Sets the retry flag that indicates if the renderer processing the event can retry a failed rendering.

IDMSourceRenderEvent

virtual IDMSourceRenderEvent& setRetry(Boolean allowRetry = true);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
---	---

allowRetry A Boolean value that indicates if the renderer can retry a failed rendering.
The default allows the renderer to retry.

setTargetInfo Sets information that is passed to the target renderer.

virtual IDMSourceRenderEvent& setTargetInfo(unsigned long info);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
---	---

info A value that represents the information to pass to the target renderer.

targetInfo Returns the target-defined information.

Note: This is an extra member the source and target can use to pass additional information to each other. You can use it to implement the specialized behavior for user-defined rendering mechanisms and formats.

virtual unsigned long targetInfo() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
--	---

targetName Returns the target render-to name. For example, the source can use this member to indicate to the target where it has placed the data it has rendered (that is, the name of a shared memory segment or file).

virtual IString targetName() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
--	---

targetWindow

Returns a pointer to the target window object.

virtual IWindow* targetWindow() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
---	---

targetWindowHandle

Returns the handle of the target window object.

virtual IWindowHandle targetWindowHandle() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y I
--	---

IDMSourceRenderEvent

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter 2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetDropEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDMEvent
        IDMTargetEvent
          IDMTargetDropEvent
  
```

Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	179	setDropPosition	181
	container	180	setTargetInfo	181
	containerId	180	targetInfo	181
	dropPosition	180	~IDMTargetDropEvent	180
	object	180		

The IDMTargetDropEvent class encapsulates a direct manipulation drop event when a user drops drag items onto a target. Objects of this class are constructed by the target handler, IDMTargetHandler (p. 195). An object of this class is constructed once when the drop occurs.

Objects of this class provide information about the drag items from the static IDMTargetOperation (p. 207) object. The target operation object is created during the processing of the initial IDMTargetEnterEvent (p. 187) for a target.

PM Objects of this class wrapper the DM_DROP message and the container notification CN_DROP.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetDropEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMTargetDropEvent

IDMTargetDropEvent(const IEvent& event);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the event object.

Exceptions	
IAccessError	The object for the target drop event was not created. The drag information is not accessible.

~IDMTargetDropEvent

virtual
~IDMTargetDropEvent();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Container Support

Use these members to provide drop support for the container drop event.

container Returns a pointer to the container where the drop occurred.

virtual IContainerControl*
container() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

containerId Returns the target container's identifier (ID).

virtual unsigned long
containerId() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

object Returns a pointer to the object over which the drop occurred. If the drop occurred over the container's white space or the window is not a container, 0 is returned.

Note: The white space of a container is an area that no container objects occupy.

virtual IContainerObject*
object() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Target Drop Information

Use these members to set and query drop and target information.

dropPosition Returns the position where the drop occurred. This position is given in desktop coordinates.

IDMTargetDropEvent

```
virtual IPoint  
    dropPosition() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setDropPosition

Sets the position where the drop occurred. This position is given in desktop coordinates.

```
virtual IDMTargetDropEvent&  
    setDropPosition( const IPoint& position );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

position A reference to the point object that indicates where the drop occurred.

setTargetInfo Sets information that is passed to the source renderer.

```
virtual IDMTargetDropEvent&  
    setTargetInfo( unsigned long info );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

info A value that represents the information to pass to the source renderer.

targetInfo Returns the target-defined information.

Note: This is an extra member the source and target can use to pass additional information to each other. You can use it to implement the specialized behavior for user-defined rendering mechanisms and formats.

```
virtual unsigned long  
    targetInfo() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDMTargetEvent		
presSpace	releasePresSpace	

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IDMTargetDropEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetEndEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDMEvent
        IDMTargetEndEvent
  
```

Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	184	renderingFailed	185
	alternateWindow	183	targetCanRetry	185
	alternateWindowHandle	184	targetInfo	185
	dragItem	184	~IDMTargetEndEvent	184

The IDMTargetEndEvent class represents objects that the source sends to the target when the source renderer completes the rendering of a drag item. Objects of this class are constructed by the target handler, IDMTargetHandler (p. 195). An object of this class is constructed for each drag item when a source renderer completes rendering.

Objects of this class provide source information about the drag item's rendering results.

Note: The User Interface Class Library does not process this event in any of its default rendering mechanisms and formats.

[PM] Objects of this class wrapper the DM_RENDERCOMPLETE message.

Public Functions

Alternate Window

Use these members to query the alternate source window. For example, the alternate source window can be used to implement source rendering on a secondary thread.

alternateWindow

Returns a pointer to the alternate source window object.

IDMTargetEndEvent

```
virtual IWindow*
    alternateWindow() const;
```

Win

PM

Motif

I

Y

I

alternateWindowHandle

Returns the handle of the alternate source window object.

```
virtual IWindowHandle
    alternateWindowHandle() const;
```

Win

PM

Motif

I

Y

I

Constructors

You can construct and destruct objects of this class.

IDMTargetEndEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMTargetEndEvent( const IEvent& event );
```

Win

PM

Motif

I

Y

I

event A reference to the event object.

Exceptions	
IInvalidRequest	The object for the target end event was not created. The drag transfer information may be invalid.
IInvalidRequest	The object for the target end event was not created. The drag item may be invalid.

~IDMTargetEndEvent

```
virtual
    ~IDMTargetEndEvent();
```

Win

PM

Motif

I

Y

I

Rendering

Use these members to provide the rendering results returned by the source.

dragItem Returns the handle of the drag item that the source has completed rendering.

```
virtual IDMItem::Handle
    dragItem() const;
```

Win

PM

Motif

I

Y

I

IDMTargetEndEvent

renderingFailed

Returns true if the source cannot perform the rendering operation. The target can accept a retry attempt, but if the target is not prepared to retry, it must call `IDMTargetRenderer::informSourceOfCompletion` (p. 215) to end the direct manipulation.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
renderingFailed() const;	<i>I</i>	<i>Y</i>	<i>I</i>

targetCanRetry

Returns true if the target can retry its part of a failed rendering. The source has successfully completed its part of the rendering. The default is false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetCanRetry() const;	<i>I</i>	<i>Y</i>	<i>I</i>

targetInfo

Returns the target-defined information.

Note: This is an extra member the source and target can use to pass additional information to each other. You can use it to implement the specialized behavior for user-defined rendering mechanisms and formats.

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetInfo() const;	<i>I</i>	<i>Y</i>	<i>I</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IDMTargetEndEvent

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetEnterEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDMEvent
        IDMTargetEvent
          IDMTargetEnterEvent
  
```

Inherited By None.

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	188	isDragAfter	188
	container	188	object	189
	containerId	188	position	189
	defaultOperation	189	setDefaultOperation	190
	dropIndicator	189	setDropIndicator	190
	isAboveFirst	188	~IDMTargetEnterEvent	188

The IDMTargetEnterEvent class encapsulates direct manipulation target enter events when a user drags drag items over a target. Objects of this class are constructed by the target handler, IDMTargetHandler (p. 195). Objects of this class are constructed for each target enter event.

Objects of this class provide information about the drag items from the static IDMTargetOperation (p. 207) object. The target operation object is created during the processing of the initial target enter event for a target.

The event result field is of particular importance for these event objects. The result indicates whether the drag item objects can be dropped on the target, and, if so, what the default drag operation is. This class provides specialized functions to set the components of the event result.

PM Objects of this class wrapper the DM_DRAGOVER message and the container notifications CN_DRAGOVER and CN_DRAGAFTER.

Public Functions

IDMTargetEnterEvent

Constructors

You can construct and destruct objects of this class.

IDMTargetEnterEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMTargetEnterEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the event object.

~IDMTargetEnterEvent

```
virtual  
~IDMTargetEnterEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Container Support

Use these members to query information for the container target enter event.

container Returns a pointer to the container within which the target enter event occurred.

```
virtual IContainerControl*  
container() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

containerId Returns the target container's identifier (ID).

```
virtual unsigned long  
containerId() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isAboveFirst Returns true if the target enter event was generated above the first list item in one of the list views of a container. An example of a container list view is details view.

```
virtual Boolean  
isAboveFirst() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isDragAfter Returns true if the target enter event was generated above the first list item, between list items, or after the last list item in one of the list views of a container. An example of a container list view is details view.

False is returned if the event was generated over a list item.

IDMTargetEnterEvent

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isDragAfter() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

object

Returns a pointer to the object over which the target enter event occurred. If the event occurred over the container's white space or the window is not a container, 0 is returned.

Note: The white space of a container is an area that no container objects occupy.

virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
object() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to set and query event information for this class.

defaultOperation

Returns the default drag operation for the target. IDMOperation::operation (p. 124) provides a list of the drag operations.

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultOperation() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

dropIndicator

Returns the drop indicator, which is one of the following:

- IDM::notOk
- IDM::ok
- IDM::operationNotOk
- IDM::neverOk

virtual DropIndicator	<u>Win</u>	<u>PM</u>	<u>Motif</u>
dropIndicator() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

position

Returns the position of the hot spot of the pointing device.

virtual IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
position() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The pointing device's position was not returned. The drag information is not accessible.

IDMTargetEnterEvent

setDefaultOperation

Sets the default drag operation for the target.

```
IDMTargetEnterEvent&  
    setDefaultOperation( unsigned long operation );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

operation A value that represents the default drag operation.
IDMOperation::operation (p. 124) provides a list of the drag operations.

setDropIndicator

Sets the drop indicator, which is one of the following:

- IDM::notOk
- IDM::ok
- IDM::operationNotOk
- IDM::neverOk

```
IDMTargetEnterEvent&  
    setDropIndicator( DropIndicator indicator );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

indicator A value that represents the drop indicator.

Inherited Public Functions

IDMTargetEvent		
presSpace	releasePresSpace	

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IDMTargetEnterEvent

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetEvent

IDMTargetEvent

Derivation

- IBase
- IVBase
- IEvent
- IDMEvent
- IDMTargetEvent

Inherited By

- IDMTargetDropEvent
- IDMTargetEnterEvent
- IDMTargetLeaveEvent

Header File idmevent.hpp

Members	Member	Page	Member	Page
	Constructor	192	releasePresSpace	193
	presSpace	193	~IDMTargetEvent	192

The IDMTargetEvent class is the common base class for the direct manipulation target event classes: IDMTargetEnterEvent (p. 187), IDMTargetLeaveEvent (p. 205), and IDMTargetDropEvent (p. 179).

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

IDMTargetEvent(const IEvent& event);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

event A reference to the event object.

~IDMTargetEvent

Releases the drawing (presentation) space if you have not explicitly released it.

IDMTargetEvent

```
virtual  
~IDMTargetEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Target Emphasis

Use these members to access and release the drawing space needed to draw target emphasis.

Note: You must use the drawing (presentation) space returned by IDMTargetEvent::presSpace (p. 193) to draw the target emphasis. The drawing space returned by IWindow::presSpace (Vol. II) will not work.

presSpace Acquires and returns the presentation space handle to use for drawing the target emphasis.

```
virtual IPresSpaceHandle  
presSpace();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
IAccessError	The presentation space was not obtained. An invalid window handle may have been specified or a direct manipulation is not in progress.	

releasePresSpace

Releases the presentation space handle that is used for drawing the target emphasis.

```
virtual void  
releasePresSpace();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
InvalidRequest	The presentation space was not released. An invalid presentation space may have been specified or a direct manipulation is not in progress.	

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IDMTargetEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetHandler

Derivation

```

IBase
IVBase
IHandler
IDMHandler
IDMTargetHandler
  
```

Inherited By None.

Header File idmtgth.hpp

Members	Member	Page	Member	Page
	Constructor	195	replaceRenderer	198
	addRenderer	197	setRenderer	198
	allocateOperation	200	targetDrop	199
	dispatchHandlerEvent	199	targetEnd	199
	findRendererFor	201	targetEnter	200
	findRenderersFor	201	targetHelp	200
	numberOfRenderers	197	targetLeave	200
	removeRenderer	197	~IDMTargetHandler	196
	renderer	197		

The IDMTargetHandler class processes events occurring on the target of a direct manipulation. You must add an object of this class to a window so it can support target direct manipulation. Objects of this class do not allow objects to be dragged from a window. See IDMSourceHandler (p. 145) for information about dragging objects from a window.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetHandler

```

1 IDMTargetHandler( IContainerControl* containerControl );

```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IDMTargetHandler

containerControl

A pointer to the container control object.

Creates an object of this class using a pointer to a container control object. The constructed handler is automatically attached to the specified container.

2	IDMTargetHandler(IWindow* window);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

window A pointer to the window object.

Creates an object of this class using a pointer to an IWindow-based object that represents a control that is not supported by the default User Interface Class Library support for direct manipulation. These controls can be system-defined controls or custom controls. The constructed handler is automatically attached to the specified window.

3	IDMTargetHandler(IEntryField* entryField);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

entryField A pointer to the entry field object.

Creates an object of this class using a pointer to an entry field object. The constructed handler is automatically attached to the specified entry field.

4	IDMTargetHandler(IMultiLineEdit* multiLineEdit);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

multiLineEdit

A pointer to the multiline edit object.

Creates an object of this class using a pointer to a multiline edit (MLE) object. The constructed handler is automatically attached to the specified MLE.

5	IDMTargetHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This is the default constructor.

~IDMTargetHandler

virtual	~IDMTargetHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

IDMTargetHandler

Renderer Processing

These members override the rendering functions defined in the base IDHandler (p. 48) class. Use them to access the IDTargetRenderer (p. 213) objects involved in a direct manipulation. Objects of this class render the drag item objects that are compatible with the rendering mechanisms and formats that the User Interface Class Library supports.

addRenderer Adds the specified renderer to the set of candidates maintained by this handler and uses it to render objects dragged from a target window where this handler is attached. The renderer is added to the end of the renderer collection.

Note: Renderers are maintained by position, which are 1-based.

<pre>virtual IDTargetHandler& addRenderer(const IDTargetRenderer& newRenderer);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

newRenderer

A reference to the renderer object.

numberOfRenderers

Returns the number of renderers associated with the target handler.

<pre>virtual unsigned numberOfRenderers();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

removeRenderer

Removes the specified renderer from this target handler.

Note: Renderers are maintained by position, which are 1-based.

<pre>virtual IDTargetHandler& removeRenderer(const IDTargetRenderer& rendererToRemove);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

rendererToRemove

A reference to the target renderer object to remove from the renderer collection.

renderer

Returns the renderer with the specified position.

Note: Renderers are maintained by position, which are 1-based.

<pre>virtual IDTargetRenderer* renderer(unsigned position);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IDMTargetHandler

position A value that represents the location of the source renderer.

replaceRenderer

Replaces a given renderer with another specified renderer.

Note: Renderers are maintained by position, which are 1-based.

```
virtual IDMTargetHandler&
    replaceRenderer( unsigned position,
                    const IDMTargetRenderer& replacement );
```

Win PM Motif
Y Y N

position A value that represents the location of the target renderer to replace in the
 renderer collection.

replacement
 A reference to the replacement renderer object.

setRenderer Sets the renderer for this handler. Removes any pre-existing target renderers.

```
virtual IDMTargetHandler&
    setRenderer( const IDMTargetRenderer& newRenderer );
```

Win PM Motif
Y Y N

newRenderer
 A reference to the target renderer object.

Inherited Public Functions

IDMHandler		
defaultDragHandler	defaultTargetHandler	enableDropOn
defaultDropTargetHandler	enableDragDropFor	setDefaultSourceHandler
defaultSourceHandler	enableDragFrom	setDefaultTargetHandler

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IDMTargetHandler

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

These members dispatch the various events for a direct manipulation target.

dispatchHandlerEvent

Overrides the default handler's dispatcher. This function processes the direct manipulation events for the target.

```
virtual Boolean
    dispatchHandlerEvent( IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the event object.

Event Processing

These members are dispatched in response to specific direct manipulation events for the target.

targetDrop Called when the drag item objects are dropped onto the target window.

```
virtual Boolean
    targetDrop( IDMTargetDropEvent& event,
               Boolean container = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target drop event object.
container A Boolean value that identifies the container as the target of the target drop event.

targetEnd Called when the source renderer completes rendering a specific drag item. The source renderer posts the target end event and this function processes it.

Note: This function is not used by any of the default renderers supported by the User Interface Class Library.

```
virtual Boolean
    targetEnd( IDMTargetEndEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMTargetHandler

event A reference to the target end event object.

targetEnter Called when the drag item objects enter the target window.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetEnter(IDMTargetEnterEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target enter event object.

targetHelp Called when the user requests help while dragging an object over the target window.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetHelp(IDMTargetHelpEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target help event object.

targetLeave Called when the drag item objects leave the target window.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetLeave(IDMTargetLeaveEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target leave event object.

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

allocateOperation

Creates a new IDMTargetOperation (p. 207) object.

This function is called when the target handler is handling the processing of a new direct manipulation. Derived classes should override this function and return a class derived from IDMTargetOperation.

This function returns a handle to the target operation object.

virtual IDMTargetOperation::Handle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
allocateOperation(IDMTargetEnterEvent& event) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target enter event object.

IDMTargetHandler

findRendererFor

Finds the appropriate target renderer for the drag item. Returns the position of the renderer or 0 if a renderer cannot be found.

```
virtual unsigned
    findRendererFor( const IDItem::Handle& item );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to the handle of the drag item object.

findRenderersFor

Finds the appropriate target renderers for the specified target operation.

```
virtual Boolean
    findRenderersFor(
        const IDMTargetOperation::Handle& targetOperation);
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

targetOperation

A reference to the handle of the target operation object.

Inherited Protected Functions

IDMHandler		
addRenderer	numberOfRenderers	renderer
isContainerControl	removeRenderer	replaceRenderer

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IDMTargetHelpEvent

IDMTargetHelpEvent

Derivation

IBase
IVBase
IEvent
IDMEvent
IDMTargetHelpEvent

Inherited By None.

Header File idmevent.hpp

Members

Member	Page
Constructor	202
operation	203
~IDMTargetHelpEvent	203

The IDMTargetHelpEvent class encapsulates direct manipulation target help events when a user requests help over a target. Objects of this class are constructed by the target handler, IDMTargetHandler (p. 195). An object of this class is constructed once when help is requested.



Objects of this class wrapper the DM_DROPHELP message and the container notification CN_DROPHELP.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetHelpEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMTargetHelpEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

event A reference to the event object.

IDMTargetHelpEvent

~IDMTargetHelpEvent

```
virtual                                Win PM Motif  
~IDMTargetHelpEvent();              I   Y   N
```

Event Information

Use these members to query event information for this class.

operation Returns the operation (for example, IDMOperation::move). Use this information to provide operation-specific help for the drag operation. IDMOperation::operation (p. 124) provides a list of the drag operations.

```
virtual unsigned long                 Win PM Motif  
operation() const;                   I   Y   N
```

Exceptions	
IAccessError	The operation was not returned. The drag information is not accessible.

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IDMTargetHelpEvent

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetLeaveEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDMEvent
        IDMTargetEvent
          IDMTargetLeaveEvent
  
```

Inherited By None.

Header File idmevent.hpp

Members	Member	Page
	Constructor	205
	~IDMTargetLeaveEvent	206

The IDMTargetLeaveEvent class encapsulates direct manipulation target leave events when a user drags drag items out of a target. Objects of this class are constructed by the target handler, IDMTargetHandler (p. 195).

Objects of this class are sent to the target when either of the following occurs:

- The user drags the objects out of the target.
- The user terminates the drag while the objects are over the target.

PM Objects of this class wrapper the DM_DRAGLEAVE message and the container notification CN_DRAGLEAVE.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetLeaveEvent

Constructs objects of this class from a generic IEvent (Vol. II) object.

```
IDMTargetLeaveEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IDMTargetLeaveEvent

event A reference to the event object.

~IDMTargetLeaveEvent

```
virtual  
~IDMTargetLeaveEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDMTargetEvent		
presSpace	releasePresSpace	

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTargetOperation

Derivation

```

IBase
  IVBase
    IRefCounted
      IDMOperation
        IDMTargetOperation
  
```

Inherited By None.

Header File idmtgtop.hpp

Members	Member	Page	Member	Page
	Constructor	207	setFirstTimeEntered	211
	dropOffset	208	setStyle	209
	dropPosition	208	setTargetOperation	211
	firstTimeEntered	210	style	209
	instanceFor	210	targetOperation	211
	isStyle	208	wasDragAfter	209
	positionRelativeTo	209	~IDMTargetOperation	208
	setDropOffset	209		

The IDMTargetOperation class represents a target operation in a direct manipulation. Objects of this class provide information about target operations. You can access this information from a drag item using `IDMItem::targetOperation` (p. 97) or the static function `IDMTargetOperation::targetOperation` (p. 211).

Objects of this class are not intended to be created by application programmers. The target event handler class, `IDMTargetHandler` (p. 195) creates them.

You can access an object of this class using `IDMTargetOperation::Handle`, which reference-counts the object.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetOperation

Construct objects of this class.

IDMTargetOperation

IDMTargetOperation(IDMTargetEnterEvent& event);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to a target enter event.

Exceptions	
IAccessError	The object for the target operation was not created. The drag information is not accessible.
IInvalidRequest	The object for the target operation was not created. The target item provider failed to provide any items.

~IDMTargetOperation

Destructs objects of this class.

Note: If a container is the target of the direct manipulation, the container refresh is enabled and the container is refreshed.

virtual

~IDMTargetOperation();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Container Support

Use these members to set and query container-specific attributes.

dropOffset Queries the current drop offset. The *drop offset* is the location where the next container object dropped would be positioned as long as the target operation's drop style is not IDM::dropPosition. The location is the offset relative to the position of the last object dropped.

ISize

dropOffset() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

dropPosition Returns the drop position for *dragItem* in container workspace coordinates and sets the offset for the next drag item according to the target operation's drop style.

IPoint

dropPosition(const IDMItem::Handle& dragItem,
IDMTargetDropEvent& event);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

dragItem A reference to the handle of a drag item.

event A reference to a target drop event.

isStyle Returns true if the specified drop-style flag is set.

IDMTargetOperation

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isStyle(DropStyle dropStyle);	<i>Y</i>	<i>Y</i>	<i>N</i>

dropStyle A value that represents the drop style.

positionRelativeTo

Returns a drag item's position relative to the target container. If the drag item handle is 0, a drop point relative to the target container's origin is returned.

IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
positionRelativeTo(const IDMItem::Handle& item, IDMTargetDropEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

item A reference to the handle of a drag item.

event A reference to a target drop event.

setDropOffset

Sets the current drop offset in window coordinates. The drop offset is the location where the next container object dropped would be positioned as long as the target operation's drop style is not IDM::dropPosition. The location is the offset relative to the position of the last object dropped.

IDMTargetOperation&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDropOffset(const ISize& newPointerOffset);	<i>Y</i>	<i>Y</i>	<i>N</i>

newPointerOffset

A reference to the new drop offset.

setStyle

Sets the drop style for the target operation.

IDMTargetOperation&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setStyle(DropStyle dropStyle);	<i>Y</i>	<i>Y</i>	<i>N</i>

dropStyle A value that represents the drop style.

style

Returns the target operation's drop style.

DropStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
style() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

wasDragAfter

IDMTargetOperation

Returns true if the target enter event was generated above the first list item, between list items, or after the last list item in one of the list views of a container. An example of a container list view is the details view.

False is returned if the event was generated over a list item.

Boolean

wasDragAfter() const;

Win

PM

Motif

Y

Y

N

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

instanceFor Returns an IDMTargetOperation object.

Note: Generally, you do not call this function. This function is called by the IDMTargetHandler (p. 195) object during the processing of a target enter event.

static IDMTargetOperation::Handle

instanceFor(IDMTargetEnterEvent& event);

Win

PM

Motif

Y

Y

N

event A reference to a target enter event.

Exceptions	
IAccessError	The handle for the target operation object was not returned. The drag information is not accessible.

Operation Services

Use these members for various operational services.

firstTimeEntered

Returns a flag indicating that a drag operation is entering the target window for the first time.

Note: Generally, you do not call this function. This function is called by the IDMTargetHandler (p. 195) object during the processing of a targe enter event.

virtual Boolean

firstTimeEntered() const;

Win

PM

Motif

Y

Y

N

IDMTargetOperation

setFirstTimeEntered

Sets a flag indicating that a drag operation is entering the target window for the first time.

```
IDMTargetOperation&
    setFirstTimeEntered( Boolean flag );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setTargetOperation

Sets an IDMTargetOperation handle during the processing of the initial target enter event.

```
static void
    setTargetOperation(
        IDMTargetOperation::Handle tgtOperation);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

tgtOperation

A handle to a target operation object.

targetOperation

Returns the IDMTargetOperation handle created during the processing of the initial target enter event.

```
static IDMTargetOperation::Handle
    targetOperation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDMOperation		
addItem	removeItem	setPosition
containerObject	replaceItem	setSource
debugSupport	setContainerObject	setSourceWindowHandle
item	setContainerRefreshOff	setTargetWindowHandle
numberOfItems	setContainerRefreshOn	source
operation	setDebugSupport	sourceWindow
position	setOperation	sourceWindowHandle

IRefCounted		
addRef	removeRef	useCount

IDMTargetOperation

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMOperation		
dragInfo	dragWasInterrupted	setDragInfo

Inherited Public Data

IDMOperation		
copy	drag	link

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Type Definitions

Handle `typedef IReference < IDMTargetOperation > Handle;`

IDMTargetOperation::Handle provides access to the IDMTargetOperation objects associated with a direct manipulation. This handle manages the references to the IDMTargetOperation object and ensures that this object is not destructed until the direct manipulation is completed.

Use Handle to reference a target operation handle within this class and IDMTargetOperation::Handle externally.

This handle provides a pointer operator, which enables the handle to be treated as a pointer to an IDMTargetOperation object.



IDMTargetRenderer

Derivation IBase
 IVBase
 IDMRenderer
 IDMTargetRenderer

Inherited By None.

Header File idmtgrn.hpp

Members	Member	Page	Member	Page
	Constructor	213	renderToName	219
	canRender	215	supportsOperation	215
	informSourceOfCompletion	215	targetRender	216
	prepareAtSource	217	targetRenderComplete	216
	renderAtSource	218	targetRenderPrepare	217
	renderComplete	218	~IDMTargetRenderer	214

The IDMTargetRenderer class provides target rendering support for a direct manipulation. Objects of this class are registered with the target handler via IDMRenderer::setDefaultTargetRenderer (p. 133), IDMTargetHandler::addRenderer (p. 197), or IDMTargetHandler::setRenderer (p. 198).

The target handler selects a best-match target renderer by calling IDMTargetHandler::findRendererFor (p. 201) for a given drag item and invokes the rendering functions of this renderer when target rendering events occur.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTargetRenderer

1	IDMTargetRenderer(const char* rmfs, const char* type = IDM::any);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>
	<i>rmfs</i>	A pointer to the RMFs string.		

IDMTargetRenderer

type A pointer to the drag item types string. The default is any drag item type, IDM::any.

Use this function to construct objects of this class by providing rendering mechanisms and formats (RMFs) and drag item types that indicate the drag items that can be rendered by this target renderer.

2	IDMTargetRenderer();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This is the default constructor.

When this constructor is used, the renderer supports any type and all of the default target RMFs supported by the User Interface Class Library:

```
(IDM::rmLibrary)x(IDM::rfProcess,IDM::rfText,IDM::rfSharedMem),  
<IDM::rmFile,IDM::rfText>
```

~IDMTargetRenderer

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IDMTargetRenderer();</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

Source and Target Rendering

Use these members to support the implementation of target rendering. Members that you can use to support source rendering are identified on an individual basis.

Source rendering is rendering that involves both the source and target after the drop has occurred. The data for rendering is not available when the drop occurs, so the target requests that the source make the data available. The request for data is issued for each drag item using IDMSourceRenderEvent (p. 174), and possibly IDMSourcePrepareEvent (p. 160). The source can notify the target when it has finished rendering each drag item using IDMTargetEndEvent (p. 183). The target informs the source when it has finished rendering each drag item via IDMSourceEndEvent (p. 142).

Note: The default source rendering support in the User Interface Class Library does not use IDMTargetEndEvent.

Target rendering, the optimal form of rendering, involves mainly the target after the drop has occurred. The data for rendering is available when the drop occurs. The target informs the source when it has finished rendering each drag item via IDMSourceEndEvent (p. 142).

IDMTargetRenderer

canRender Returns true if the target renderer can render drag items with the specified attributes. This function is called for each drag item during target enter event processing.

Note: This function can be used for either source or target rendering.

```
virtual DropIndicator  
    canRender( const IDMIItem::Handle& dragItem );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

dragItem A reference to the handle of a drag item.

informSourceOfCompletion

Informs the source that the target has finished processing the data associated with the dropped item. The render completion code indicates if the target renderer successfully processed the data or if it failed.

Note: This function must be used for both source and target rendering.

```
virtual IDMTargetRenderer&  
    informSourceOfCompletion(  
        const IDMIItem::Handle& dragItem,  
        IDM::RenderCompletion code = IDM::targetSuccessful );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

dragItem A reference to the handle of a drag item.

code A value that represents the render completion code. The default is IDM::targetSuccessful.

Exceptions	
InvalidRequest	The source could not be informed that the target finished processing. The drag item is invalid.

supportsOperation

Determines if the default drag operation, which is set by the source, is supported by the drag item. If the default operation is IDMOperation::drag (p. 130), the target must select an operation. If no operation is selected, the User Interface Class Library's default selection is IDMOperation::move (p. 130), based upon Common User Access (CUA) guidelines.

If the default operation defined by the source is a value greater than IDMOperation::unknown (p. 130), the application must define the functionality of this operation and process the operation accordingly.

When the default operation defined by the source is one of the following, the default processing verifies if the drag item supports the operation:

IDMOperation::move (p. 130)
IDMOperation::copy (p. 130)

IDMTargetRenderer

IDMOperation::link (p. 130)

This function is called for each drag item during target enter event processing.

Note: This function can be used for either source or target rendering.

```
virtual Boolean  
    supportsOperation( const IDMItem::Handle& dragItem );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

dragItem A reference to the handle of a drag item.

targetRender Called when a drop event occurs for a drag item and the renderer must render the drag item. Use the rendering mechanism and format (RMF) to determine the type of rendering to implement.

This function is called for each drag item during target drop event processing.

Note: This function can be used for either source or target rendering.

```
virtual IDMTargetRenderer&  
    targetRender( IDMTargetDropEvent& event,  
                 const IDMItem::Handle& dragItem );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

event A reference to the target drop event object.

dragItem A reference to the handle of a drag item.

Exceptions	
InvalidRequest	The drag item was not rendered. The IDM::rfProcess rendering format can only be used if the source and the target are in the same process.
InvalidRequest	The drag item was not rendered. Source rendering of the drag item failed.
InvalidRequest	The drag item was not rendered. The rendering format is invalid.
InvalidRequest	The drag item was not rendered. The rendering mechanism is invalid.

targetRenderComplete

Called when the source has completed the rendering of a drag item. This function is called for each drag item during target end event processing.

This function is not used by any of the default renderers supported by the User Interface Class Library.

Note: This function is used for source rendering only.

```
virtual IDMTargetRenderer&  
    targetRenderComplete( IDMTargetEndEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>I</i>

IDMTargetRenderer

event A reference to the target end event object.

targetRenderPrepare

Called when the source requests that the target issue a source prepare event for a specific RMF.

Note: This function is used for source rendering only.

```
virtual IDMTargetRenderer&
    targetRenderPrepare( IDMTargetDropEvent& event,           Win PM Motif
                        const IDMIItem::Handle& dragItem );    I   Y   I
```

event A reference to the target drop event object.

dragItem A reference to the handle of a drag item.

Inherited Public Functions

IDMRenderer		
defaultSourceRenderer	setDefaultSourceRenderer	setSupportedRMFs
defaultTargetRenderer	setDefaultTargetRenderer	setSupportedTypes

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Renderer Specifics

Use these members to process the different rendering operations that are specific to the supported rendering mechanisms and formats.

prepareAtSource

Sends a source prepare event to the source to request source preparation for the specified drag item.

IDMTargetRenderer

The User Interface Class Library uses this function to implement the support for the <IDM::rmLibrary,IDM::rfSharedMem> rendering mechanism and format.

```
virtual Boolean                                     Win PM Motif  
    prepareAtSource( IDMTargetDropEvent& event,      Y  Y  I  
                    const IDMItem::Handle& dragItem );
```

event A reference to the target drop event object.

dragItem A reference to the handle of a drag item.

renderAtSource

Sends a source render event to the source to request source rendering for the specified drag item.

The User Interface Class Library uses this function to implement the support for the <IDM::rmLibrary,IDM::rfSharedMem> rendering mechanism and format.

```
virtual Boolean                                     Win PM Motif  
    renderAtSource( IDMTargetDropEvent& event,      Y  Y  I  
                   const IDMItem::Handle& dragItem );
```

event A reference to the target drop event object.

dragItem A reference to the handle of a drag item.

Exceptions	
IAccessError	The render event was not sent to the source. The source window handle may be invalid.
IOutOfMemory	The render event was not sent to the source. The system was unable to allocate the required memory.
IAccessError	The render event was not sent to the source. The target process could not give the source process access to the shared memory buffer because the system may be out of memory or the buffer may be locked.
IAccessError	The render event was not sent to the source. The source reported an error during the processing of the source render event.

renderComplete

Called to process a render completion event for the specified drag item.

The User Interface Class Library uses this function to implement the support for the <IDM::rmLibrary,IDM::rfSharedMem> rendering mechanism and format.

```
virtual Boolean                                     Win PM Motif  
    renderComplete( const IDMItem::Handle& dragItem, Y  Y  I  
                   void* pBuffer );
```

IDMTargetRenderrer

dragItem A reference to the handle of a drag item.
pBuffer A pointer to the shared memory buffer that contains the rendered data.

renderToName

Provides the name of the location of the rendered data. This name could be a file name or a shared memory buffer.

virtual IString
renderToName(const IDItem::Handle& dragItem);

Win
Y

PM
Y

Motif
N

dragItem A reference to the handle of a drag item.

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDMTBarButtonItem

IDMTBarButtonItem



Inherited By None.

Header File idmtbbit.hpp

Members	Member	Page	Member	Page
	Constructor	221	supportedOperationsFor	224
	generateSourceItems	223	targetDrop	221
	sourceDiscard	222	targetToolBar	223
	sourceToolBar	223	targetToolBarButton	223
	sourceToolBarButton	223	~IDMTBarButtonItem	221

The IDMTBarButtonItem class represents tool-bar-button-specific items in a direct manipulation. Tool bars create objects of the IDMTBarButtonItem class when one of the following occurs:

- A direct manipulation begins in a source tool bar button.
- A tool bar button item is dragged over a tool bar or another tool bar button during a direct manipulation.
- A menu drag item is dragged over a tool bar or tool bar button during a direct manipulation.

This class provides virtual functions that implement direct manipulation support for tool bar buttons. Derive item classes to support the direct manipulation of tool bar button items when the default User Interface Class Library support does not meet your requirements.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMTBarButtonItem

IDMTBarButtonItem

	IDMTBarButtonItem(IDMSourceOperation* sourceOperation);	Win	PM	Motif
		<i>Y</i>	<i>Y</i>	<i>I</i>

sourceOperation

A pointer to the drag source operation object to which this menu drag item object is to be added.

Use this constructor to construct tool bar button items for the source of a direct manipulation. It is called by the member function, `IDMTBarButtonItem::generateSourceItems` (p. 223).

2	<code>IDMTBarButtonItem(const IDMItem::Handle& dragItem);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

<i>dragItem</i>	A reference to a handle of a generic drag item that is created by the User Interface Class Library.
-----------------	---

Use this constructor to construct tool bar button items for the target of a direct manipulation. It is called by the member function, `IDMItemProviderFor::provideTargetItemFor` (p. 108), of the tool bar button's item provider for the tool bar button item class.

~IDMTBarButtonItem

<code>virtual</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IDMTBarButtonItem();</code>	<i>Y</i>	<i>Y</i>	<i>I</i>

Drop Processing

Use these members during the drop operation of a direct manipulation.

targetDrop	Implements tool-bar-button-item-specific rendering when the drag item that represents a tool bar button object, a menu item object, or a bitmap file object is dropped on a tool bar button. This function processes the drop in one of the following ways:
-------------------	---

- If a tool bar button is dropped on another tool bar button, it is moved either before or after the tool bar button where the drop occurred. The positioning depends upon the location of the mouse pointer. If the mouse pointer is over the left-half or top-half of the target button, the button being dragged is moved before the target button. If it is over the middle, right-half, or bottom-half of the target button, the button being dragged is moved after the target button.

IDMTBarButtonItem

- If the drag was initiated from a drop down menu, a new tool bar button is created based upon the bitmap, text, or bitmap and text in the menu drag item. The same positioning applies that is discussed in the previous paragraph.
- If a bitmap file is dropped on a tool bar button, the button's bitmap is set using the bitmap stored within the bitmap file.

Derived classes should override this function when the default support does not meet your requirements.

Note: Each tool bar button, within the context of a tool bar, must have a unique identifier (ID). The drop is rejected if a duplicate ID is detected within the same tool bar.

```
virtual Boolean  
    targetDrop( IDMTargetDropEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

event A reference to the target drop event.

Shredder Support

Use these members for tool bar button deletion when a tool bar button item is dropped on the system shredder.

sourceDiscard

Implements tool-bar-button-item-specific rendering when the operating system issues a discard request.

Derived classes should override this function when the default support does not meet your requirements.

```
virtual Boolean  
    sourceDiscard( IDMSourceDiscardEvent& event );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>I</i>

event A reference to the source discard event.



This function deletes the tool bar button object that is dropped on the Workplace Shell shredder object.

Source and Target Tool Bar

Use these members to query the source and target tool bar and the tool bar buttons.

IDMTBarButtonItem

sourceToolBar

Returns the tool bar that contains the source tool bar button.

virtual IToolBar*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceToolBar() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

sourceToolBarButton

Returns the source tool bar button where the direct manipulation started.

virtual IToolBarButton*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceToolBarButton() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

targetToolBar

Returns the tool bar that contains the target tool bar button.

virtual IToolBar*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetToolBar() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

targetToolBarButton

Returns the target tool bar button where the drop occurred.

virtual IToolBarButton*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
targetToolBarButton() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

Source Items

Use these members to access and manipulate source items involved in a direct manipulation.

generateSourceItems

Generates a tool bar button item that represents a tool bar button item object.

This function is called by the tool bar button's implementation of IDMIItemProvider::provideSourceItems (p. 105) in the template class IDMIItemProviderFor (p. 107).

The default implementation of this function in this class creates IDMTBarButtonItem objects and adds them to the source operation. If you use the IDMIItemProviderFor template class and you have defined a derived item class, you must implement IDMTBarButtonItem::generateSourceItems in your derived tool bar button item class. In your implementation, create an object of your derived tool bar button item class and call IDMOperation::addItem (p. 123) to add the objects to the source operation.

IDMTBarButtonItem

The default drag operation for tool bar button items is IDMIItem::moveable (p. 100).

Note: Use the *sourceOperation* parameter to access IDMOperation::addItem.

```
static Boolean  
generateSourceItems( IDMSourceOperation* sourceOperation );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

sourceOperation

A pointer to the source operation object.

Tool Bar Button Operations

Use these members to set the supported direct manipulation operations for this class.

supportedOperationsFor

Returns IDMIItem::moveable (p. 100) as the only operation the tool bar button item supports.

```
virtual unsigned long  
supportedOperationsFor( const IString& selectedRMFs ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

selectedRMFs

A reference to the selected RMFs for the drag item.

Inherited Public Functions

IDMIItem		
addRMF	nativeRF	setRequiresPreparation
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName

IDMTBarButtonItem

IDMItem		
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

IDMTBarButtonItem

Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IBase		
recoverable	unrecoverable	



IDMToolBarItem

Derivation

```

IBase
  IVBase
    IRefCounted
      IDMItem
        IDMToolBarItem
  
```

Inherited By None.

Header File idmtbrit.hpp

Members	Member	Page
	Constructor	227
	targetDrop	228
	~IDMToolBarItem	228

The IDMToolBarItem class represents tool-bar-specific items in a direct manipulation. Tool bars create objects of the IDMToolBarItem class when either of the following occurs:

- A tool bar button item is dragged over a tool bar during a direct manipulation
- A menu drag item is dragged over a tool bar during a direct manipulation

This class provides virtual functions that implement direct manipulation support for tool bars. Derive item classes to support the direct manipulation of tool bar items when the default User Interface Class Library support does not meet your requirements.

Note: This class does not support tool bars as the source of a direct manipulation.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDMToolBarItem

Constructs tool bar items for the target of a direct manipulation. It is called by the member function,

IDMItemProviderFor::provideTargetItemFor (p. 108), of the tool bar's item provider for the tool bar item class.

IDMToolBarItem

IDMToolBarItem(const IDMItem::Handle& dragItem);

Win

PM

Motif

Y

Y

I

dragItem

A reference to a handle of a generic drag item that is created by the User Interface Class Library.

~IDMToolBarItem

virtual
~IDMToolBarItem();

Win

PM

Motif

Y

Y

I

Drop Processing

Use these members during the drop operation of a direct manipulation.

targetDrop

Implements tool-bar-item-specific rendering when the drag item that represents a tool bar button or a menu item object is dropped on a tool bar. This function processes the drop in one of the following ways:

- If the drag was initiated from a drop down menu, a new tool bar button is created based upon the bitmap, text, or bitmap and text in the drag object. The button is placed at the end of the tool bar.
- If an existing tool bar button is dropped on a tool bar, it is treated as a move and is placed at the end of the tool bar.

Derived classes should override this function when the default support does not meet your requirements.

Note: Each tool bar button, within the context of a tool bar, must have a unique identifier (ID). The drop is rejected if a duplicate ID is detected within the same tool bar.

virtual Boolean
targetDrop(IDMTargetDropEvent& event);

Win

PM

Motif

Y

Y

I

event

A reference to the target drop event.

Inherited Public Functions

IDMItem		
addRMF	nativeRF	setRequiresPreparation

IDMToolBarItem

IDMItem		
addType	nativeRM	setRMFs
appendRMF	nativeRMF	setSelectedRMF
attributes	object	setSourceName
canBeCopied	operator =	setSourceWindowHandle
canBeLinked	removeRMF	setTargetName
canBeMoved	removeType	setTrueType
compressedRMFs	renderer	setTypes
containerName	requiresPreparation	sourceDiscard
contents	rfForThisProcess	sourceEnd
contentsSize	rfFrom	sourceItemFor
deleteRMF	rmfFrom	sourceName
dropStatus	rmFrom	sourceOperation
enableCopy	rmfs	sourcePrepare
enableLink	rmfsFrom	sourcePrint
enableMove	selectedRMF	sourceRender
generateSourceItems	setContainer	sourceWindow
hasImage	setContainerName	sourceWindowHandle
hasType	setContents	supportedOperations
image	setDropStatus	supportedOperationsFor
imageOffset	setGroup	supportsRMF
isContainer	setImage	targetDrop
isGroup	setNativeRMF	targetEnd
isOnRemovableMedia	setObject	targetName
isOpen	setOnRemovableMedia	targetOperation
isReference	setOpen	tokenForWPSObject
isTargetTheSource	setReference	trueType
matchingRMFs	setRenderer	types

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IDMToolBarItem

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDMItem		
generateSourceName		

Inherited Public Data

IDMItem		
container	linkable	open
copyable	moveable	prepare
group	none	reference

Inherited Protected Data

IDMItem		
strContents		

IBase		
recoverable	unrecoverable	



IDocumentStorage

Derivation IBase
 IVBase
 IDocumentStorage

Inherited By IFlatFileStorage
 IStructuredStorage

Header File idocstor.hpp

Members	Member	Page	Member	Page
	Constructor	234	saveFile	232
	component	232	setFileName	233
	fileName	233	~IDocumentStorage	233
	loadFile	231		

The IDocumentStorage class is an abstract base class for saving and loading components to and from files. Two concrete subclasses, IStructuredStorage and IFlatFileStorage, derive from IDocumentStorage. The document storage object is created and owned by the component (that is, an object of class IComponent).

See IFlatFileStorage (p. 250) and IStructuredStorage (p. 711) for more information on the concrete storage classes. See IComponent (p. 7) for more information about components.

Public Functions

File I/O

Use these members to save a model to, or load a model from, a file.

loadFile Loads a model from the specified file. The component deletes any existing model and replaces it with the loaded model. See IComponent::adoptModel (p. 8) for more information.

If the file cannot be loaded, the stationery creates a new, empty, model for the component to adopt. See IComponentStationery::createModel (p. 15) for more information.

IDocumentStorage

This pure virtual function is implemented in the concrete subclasses IFlatFileStorage and IStructuredStorage. See IFlatFileStorage (p. 250) and IStructuredStorage (p. 711) for more information about the storage classes.

```
virtual void  
    loadFile( const IString& name ) = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IString& The name of the file to be loaded.

saveFile

This function saves the document model to the specified file. The second and third parameters are used to indicate the type of save required, that is, **File ► Save**, **File ► Save As**, and **File ► Save Copy As**.

sameAsLoad	remember	operation
true	N/A	Save
false	true	Save As
false	false	Save Copy As

This pure virtual function is implemented in the concrete subclasses IFlatFileStorage and IStructuredStorage. See IFlatFileStorage (p. 250) and IStructuredStorage (p. 711) for more information on the storage classes.

```
virtual void  
    saveFile( const IString& name,  
              Boolean sameAsLoad,  
              Boolean remember ) = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

IString& The name of the file to be saved.
Boolean Indicates whether the name being used is the same as that used during loading (**File ► Save**), or not (**File ► Save As** and **File ► Save Copy As**).
Boolean Indicates that the document's file name should be changed to the supplied name.

Informational

Use these members to obtain information about the document.

component Provides the document component that owns this document storage object.

IDocumentStorage

```
IComponent&  
    component() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

fileName Returns the current file name for the document. If no file is currently associated with the document, a null string (" ") is returned. The file name is initially null.

```
const IString&  
    fileName() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Constructors

You cannot construct, destruct, or copy an object of this class directly.

~IDocumentStorage

```
virtual  
    ~IDocumentStorage();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Title Components

Use this member to set the title of a component.

setFileName Sets the path and file name for the document.

Sets the document name to the specified name, without the path information. See `IComponent::setDocumentName` (p. 11) for more information.

If the specified name is a null string (" "), the document name becomes (Untitled).

Called by the framework when the end-user selects **File ► SaveAs**, **File ► Open**, or **File ► New**.

```
virtual void  
    setFileName( const IString& name );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IString& The new file name for the document.

Inherited Public Functions

IDocumentStorage

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You cannot construct, destruct, or copy an object of this class directly.

IDocumentStorage

1	<code>IDocumentStorage(IComponent&);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IComponent&
The component that will own the new storage object.

See IComponent (p. 7) for more information about components.

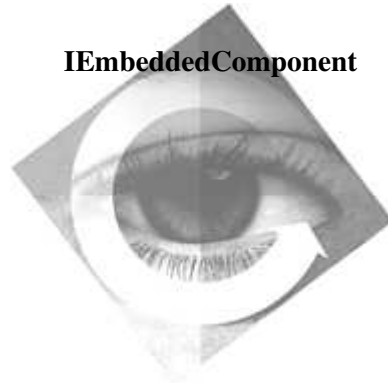
2	<code>IDocumentStorage(const IDocumentStorage&);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

This function is the copy constructor for IDocumentStorage.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IEmbeddedComponent



IEmbeddedComponent

Derivation IMGrabbable
IEmbeddedComponent

Inherited By None.

Header File iembmod.hpp

Members	Member	Page	Member	Page
	Constructor	236	name	238
	area	237	number	238
	aspect	236	objectArea	239
	bundle	238	operator <<=	241
	doDragDrop	242	operator >>=	241
	draw	239	rect	242
	goActive	236	resize	240
	goInactive	237	setArea	241
	grabHandles	238	setAspect	237
	initialize	239	setObjectArea	241
	inWindow	242	setRect	242
	isInitialized	238	showGrabHandles	240
	isInPlaceActive	238	~IEmbeddedComponent	236

Objects of this class represent server components that are embedded in a container component. A container's embedder model (that is, an object derived from IEmbedderModel) maintains a list of IEmbeddedComponent objects, where each embedded component acts as a surrogate for an actual OLE server component (that lives in another address space). The embedded component manages the container's perspective of the contained object, that is, its allocated area within the container's window. Embedded components are created and owned by embedder models. See IEmbedderModel (p. 243) for more information.

Public Functions

Constructors

You can construct and destruct objects of this class. In this release, you cannot copy objects of this class.

IEmbeddedComponent

IEmbeddedComponent

Embedded components are constructed by the container component, using the factory function `IEmbedderModel::createEmbeddedComponent` (p. 245). If you derive a class from `IEmbeddedComponent`, the derived class's constructor must explicitly or implicitly call this default constructor.

1	<code>IEmbeddedComponent();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

This function creates an `IEmbeddedComponent` object. It is called by `IEmbedderModel::createEmbeddedComponent` (p. 245).

2	<code>IEmbeddedComponent(const IEmbeddedComponent&);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

Do **not** use this function. Copying of embedded components is not supported in the current release. This function only exists to support the Extended Type system and is otherwise unused. Calls to this function directly, or using the dynamic copy function `::copy`, result in a run-time assertion.

~IEmbeddedComponent

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IEmbeddedComponent();</code>		<i>Y</i>	<i>N</i>	<i>N</i>

Embedded Component Management

Use these members to manage embedded components.

aspect Returns the current aspect of the embedded component. Currently, only values of `kContent` and `kIcon` are supported. By default, the aspect will be “`kContent`” unless “`setAspect`” has been called.

<code>IComponent::EAspect</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>aspect() const;</code>	<i>Y</i>	<i>N</i>	<i>N</i>

goActive Activates the embedded component by invoking the specified operation (or verb) on the associated server object. The framework ensures that the server application is running, and starts it if necessary.

The framework calls this function when the end-user double clicks on an embedded component (invoking the default operation for the application), or when the end-user selects an embedded component and then selects an operation from the **Edit** menu or

IEmbeddedComponent

the object's popup menu. The function passes the selected operation (verb) to the application.

See IEmbedderModel::selectComponent (p. 246) for more information.

```
void
goActive( long lVerb = 0 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Exceptions	
IAccessError	Failed to activate server. Recoverable.

goInactive Deactivates an embedded component, if it is active. This function is called by the framework when deselecting a component. See IEmbedderModel::selectComponent (p. 246) for more information.

```
void
goInactive();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

setAspect Sets the desired aspect of the component to the specified aspect. In the current release, only values **kContent** and **kIcon** are supported. Calls to setAspect with other values are ignored.

This function only takes effect if called before the object is initialized. Dynamic aspect changes are not supported at this time. You can override IEmbeddedComponent::initialize to call this function before initialization, but be sure to call the base initialize after the aspect is set.

```
void
setAspect( const IComponent::EAspect );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IComponent::EAspect
The aspect (kContent or kIcon) to be set as the current aspect.

Informational

Use these members to obtain information about an embedded component.

area Returns the area allocated for the embedded component in the container's view window.

IEmbeddedComponent

	<code>const IRectangle& area() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
bundle	Returns the IGUIBundle object for the server application that this embedded component represents.			
	<code>IGUIBundle& bundle() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
grabHandles	Returns the grab handles for the embedded component if there are any, otherwise returns NULL.			
	<code>IGrabHandles* grabHandles() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
isInitialized	Returns true if the embedded component is initialized (that is, IEmbeddedComponent::initialize has been called), otherwise returns false.			
	<code>Boolean isInitialized() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
isInPlaceActive	Returns true if the embedded component is currently in-place active, otherwise returns false.			
	<code>Boolean isInPlaceActive() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
name	Returns the sub-storage name for this embedded component in the container component's structured storage file. The name is of the format EmbeddedComponent< <i>number</i> >, where <i>number</i> is the embedded component's number (see IEmbeddedComponent::number). You can override this function in a subclass to provide a different substorage naming convention. Called by the framework when the component is first embedded in a container.			
	<code>virtual IString name() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
number	Returns the embedded component number. This number is used by the framework to provide unique sub-storage names for each embedded component (see IEmbeddedComponent::name).			

IEmbeddedComponent

The container object (derived from IEmbedderModel) manages a list of embedded components. The number of a component is based on its order in this list: the number of a new embedded component is one greater than the number of the last embedded component in the list. New components are always added to the end of the list. See IEmbedderModel::adopt (p. 244) for more information.

unsigned	<u>Win</u>	<u>PM</u>	<u>Motif</u>
number() const;	<i>Y</i>	<i>N</i>	<i>N</i>

objectArea Returns the server object's requested area (extent).

virtual IRectangle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
objectArea() const;	<i>Y</i>	<i>N</i>	<i>N</i>

Initialization

Use this member to initialize the embedded component, once it has been added.

initialize Performs various initialization functions on an IEmbeddedComponent object once it is added to a container. Called by the framework after a new component has been created (for example, paste from clipboard, drag and drop, stream in, or create new). The framework constructs the object, inserts it into the container, and finally initializes it.

You can override this function in a subclass to perform additional application-specific initialization, but your override must still call this base class function.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
initialize(Boolean,	<i>Y</i>	<i>N</i>	<i>N</i>
Boolean);			

Painting

Use these members to draw the embedded component, and control the display of the move/resize handles of the embedded component.

draw This function is called on each embedded component by IView::draw, which, in turn, calls out to the server and asks them to draw. You should call this function if you are overriding IView::draw, and pass the parameters from IView::draw right through.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
draw(IPresSpaceHandle&,	<i>Y</i>	<i>N</i>	<i>N</i>
const IRectangle&,			
Boolean metaFile = false);			

IEmbeddedComponent

The parameters for this function are as follows:

IPresSpaceHandle&

The presentation space handle.

IRectangle&

Defines the space in which the component should be drawn (if the next parameter is true), or represents the invalid area in the window (if the next parameter is false).

Boolean Indicates that a metafile representation should be drawn for the embedded component.

showGrabHandles

Displays or removes the move/resize handles for the embedded component. If the parameter is true, this function constructs an instance of IGrabHandles for the component. If the parameter is false, the grab handles (if present) are destroyed. Grab handles, once constructed, take over mouse event handling for their enclosed area, providing resizing and drag-and-drop support.

The framework currently does not support grab handles showing on more than one embedded component at a time.

See IGrabHandles (p. 339) and IMGrabbable (p. 437) for more details.

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
showGrabHandles(Boolean bShow = true);	Y	N	N

The parameter for this function is as follows:

Boolean Indicates whether to display or to remove the grab handles.

Sizing Notification Handlers

Use these members to set the size of an embedded component.

resize

Resizes the embedded component, as follows:

1. Changes the allocated area within the container, using setArea.
2. Notifies the server of the change, using setObjectArea.
3. Checks if the grab handles are currently showing, using grabHandles.
4. Updates the grab handles if they are showing, using changeTo.

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resize(const IRectangle&);	Y	N	N

The parameter for this function is as follows:

IEmbeddedComponent

IRectangle&

The area in which the embedded component is placed.

setArea

Sets the area allocated for the embedded component in the container's view window.

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setArea(const IRectangle&);	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IRectangle&

The area to be allocated for the embedded component.

setObjectArea

Notifies the server object of an area (extent) change. If the server ignores the notification, the representation of the component within the container is scaled to fit the allocated area.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setObjectArea(const IRectangle&);	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IRectangle&

The new area.

Streaming

Use these members to stream embedded components in and out.

operator <<= The stream-in operator for objects of this class.

IBaseStream&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator <<=(IBaseStream& fromwhere);	<i>Y</i>	<i>N</i>	<i>N</i>

operator >>= The stream-out operator for objects of this class.

IBaseStream&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator >>=(IBaseStream& towhere) const;	<i>Y</i>	<i>N</i>	<i>N</i>

IEmbeddedComponent

Inherited Public Functions

IMGrabbable		
doDragDrop	inWindow	rect

Protected Functions

Overrides of IMGrabbable Members

These members override the equivalent functions of IMGrabbable (p. 437).

doDragDrop Overrides the doDragDrop function of IMGrabbable. See IMGrabbable::doDragDrop (p. 437) for more information.

virtual Boolean
doDragDrop(const IPoint&);

Win
Y

PM
N

Motif
N

inWindow Overrides the inWindow function of IMGrabbable. See IMGrabbable::inWindow (p. 438) for more information.

virtual IWindow&
inWindow() const;

Win
Y

PM
N

Motif
N

rect Overrides the rect function of IMGrabbable. See IMGrabbable::rect (p. 438) for more information.

virtual IRectangle
rect() const;

Win
Y

PM
N

Motif
N

setRect Overrides the setRect function of IMGrabbable. See IMGrabbable::setRect (p. 438) for more information.

virtual void
setRect(const IRectangle&);

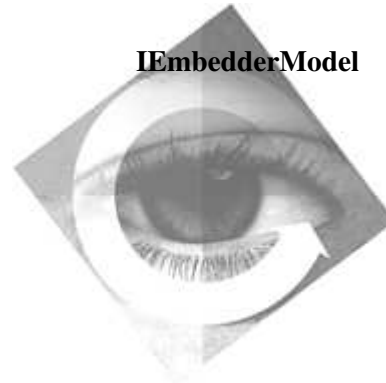
Win
Y

PM
N

Motif
N

The parameter for this function is as follows:

IRectangle&
The new location for the grabbable object.



IEmbedderModel

Derivation

```

IBase
  IVBase
    IModel
      IEmbedderModel
  
```

Inherited By None.

Header File iembmod.hpp

Members	Member	Page	Member	Page
	Constructor	244	operator <<=	247
	adopt	244	operator >>=	248
	attachTo	247	orphan	245
	componentList	245	selectComponent	246
	createEmbeddedComponent	245	selectedComponent	247
	deleteAll	245	~IEmbedderModel	244
	hasChanged	247		

Use the IEmbedderModel class to provide models for container components. Container components have all the functionality of server components (that is, they can be embedded), and in addition they can have other components embedded in them.

Create your own derived class of IEmbedderModel to create the model for a container. The two key member functions you use when customising the framework are selectedComponent and componentList. To manage its embedded components, the IEmbedderModel object keeps a list of IEmbeddedComponent objects, which it can use to select specific embedded components.

IEmbedderModel objects use structured storage (through the IStructuredStorage class) since they need to be able to store one or more embedded components.

See IEmbeddedComponent (p. 235) for more information about embedded components and see IStructuredStorage (p. 711) for more information about structured storage.

Public Functions

IEmbedderModel

Constructors

You can construct and destruct objects of this class. In this release, you cannot copy objects of this class.

IEmbedderModel

1	<code>IEmbedderModel();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

Model objects are constructed by the framework's factory function `IComponentStationery::createModel`. If you provide a model derived class, the derived class's constructor must explicitly or implicitly call this default constructor.

2	<code>IEmbedderModel(const IEmbedderModel&);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>N</i>	<i>N</i>

Do **not** use this function. Copying of embedder models is not supported in the current release. This function only exists to support the Extended Type system and is otherwise unused. Calls to this function directly, or using the dynamic copy function `::copy`, will result in a run-time assertion.

~IEmbedderModel

<code>virtual</code> <code>~IEmbedderModel();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

Embedded Component Management

Use these members to manage embedded components. See `IEmbeddedComponent` (p. 235) for more information about embedded components.

adopt

Adopts a newly created component and adds it to the container. New components are added to the end of the container's list of embedded components.

Called by the framework immediately after the construction of a new embedded component. The new component is not initialized at this point.

<code>virtual void</code> <code>adopt(IEmbeddedComponent*);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IEmbedderModel

*IEmbeddedComponent**

A pointer to the embedded component to be adopted.

componentList

Returns the list of embedded components. The returned list is of type IEmbedderModel::ComponentList, and can be queried and iterated over using the standard Open Class Collection public interfaces.

```
const ComponentList&
  componentList() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

createEmbeddedComponent

Factory function for creating IEmbeddedComponent objects. Embedded components represent servers that are embedded in a container. Derived classes can override this function to create customized embedded components.

```
virtual IEmbeddedComponent*
  createEmbeddedComponent() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

deleteAll

Closes and deletes all the embedded components in a container, and then deletes the model. The embedded components are not removed from the persistent storage. The purpose of this function is to shut down, as opposed to remove, embedded components. The framework calls this function when the server application is being terminated.

```
virtual void
  deleteAll();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

orphan

Closes and removes the specified embedded component from the container and returns the orphaned component. This function also removes the embedded component from the persistent storage.

Called by the framework when an embedded component is removed from the container, for example, during a clipboard **Cut** operation.

```
virtual IEmbeddedComponent*
  orphan( IEmbeddedComponent& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IEmbeddedComponent&

A reference to the embedded component that is to be closed and removed.

IEmbedderModel

selectComponent

Selects the specified embedded component. Selecting a component has the following effects:

- The component becomes the target of the framework-provided default **Cut**, **Copy** and **Delete** commands.
- Move/resize handles appear on the selected component. See `IEmbeddedComponent::showGrabHandles` (p. 240).
- The previously selected component, if any, is deselected.

By default, the framework selects a component in the following situations:

- When the end-user clicks or double-clicks on an embedded component.
- When the end-user inserts a newly created object into the container, using the **Insert Object** command.

You can deselect a component by calling `selectComponent(NULL)`, that is, by passing a null pointer to this function. A component becomes deselected in the following situations:

- Another component is selected.
- The end-user clicks over an area where there is no component.
- The selected component is removed from the container, for example, during a clipboard **Cut** operation. See `IEmbedderModel::orphan`.

When a given component is deselected, the following occurs:

- The component's resize handles are removed, `IEmbeddedComponent::showGrabHandles(false)` (p. 240).
- The component, if activated, is deactivated, using `IEmbeddedComponent::goInactive` (p. 237).

See `IEmbeddedComponent` (p. 235) for more information about embedded components.

```
virtual void  
    selectComponent( IEmbeddedComponent* );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

*IEmbeddedComponent**

A pointer to the embedded component to be selected or `NULL` to deselect the current component.

IEmbedderModel

selectedComponent

Returns the currently selected embedded component, or NULL, if no component is selected. Components are selected using the selectComponent function.

IEmbeddedComponent*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
selectedComponent() const;	<i>Y</i>	<i>N</i>	<i>N</i>

Overrides of IModel Functions

These members override the equivalent functions in IModel (p. 703).

attachTo

Attaches the model to a component. This function is called by IComponent::adoptModel (p. 8) when the component adopts the model. This function performs some container-specific initialization, and calls IModel::attachTo (p. 706).

See IComponent (p. 7) for more information about components.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
attachTo(IComponent&);	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IComponent&
A reference to a component.

hasChanged

Returns whether or not the model, or any of its embedded components, have changed. This function overrides the base class IModel::hasChanged (p. 704) function to include changes to embedded components.

Applications can call IModel::hasChanged directly to determine the change state of the native model data, without considering its embedded components. For example, if an embedded component has changed but the container model has not, IModel::hasChanged returns false, and IEmbeddedComponent::hasChanged returns true.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasChanged() const;	<i>Y</i>	<i>N</i>	<i>N</i>

Streaming

Use these members to stream objects of this class in and out.

operator <<= The stream-in operator for objects of this class. This function first calls the base class (IModel) stream-in operator and then streams in the list of embedded components.

IEmbedderModel

```
virtual IBaseStream&  
operator <<=( IBaseStream& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

operator >>= The stream-out operator for objects of this class. This function first calls the base class (IModel) stream-out operator and then streams out the list of embedded components.

```
virtual IBaseStream&  
operator >>=( IBaseStream& ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

IModel		
attachTo	notifier	operator >>=
component	notifyOfChange	operator INotifier &
hasChanged	operator <<=	setChanged

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

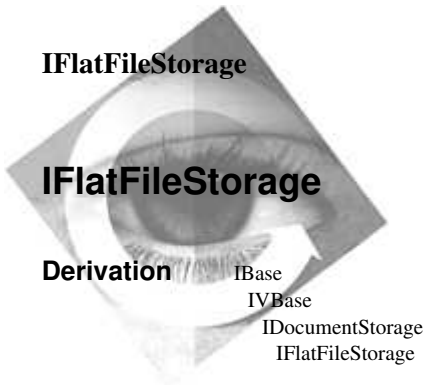
Nested Type Definitions

ComponentList

```
typedef ISequence < IElemPointer < IEmbeddedComponent > > ComponentList;
```

IEmbedderModel

The `ComponentList` type is a sequence of persistent pointers to embedded components. See the Class Library User's Guide (Compound Document Framework section), and the Compound Document Framework Sample 5 for more information.



Inherited By None.

Header File idocstor.hpp

Members				
Member	Page	Member	Page	
Constructor	250	saveFile	251	
loadFile	251	~IFlatFileStorage	251	

The framework uses this class to to manage the document storage for non-containers, that is, components whose model class derives from IModel (p. 703), as opposed to IEmbedderModel (p. 243). IFlatFileStorage objects store the document model as a flat file. Because the file being saved is not a container, it does not need to be saved using OLE structured storage.

If desired, you can force a server to use structured storage by overriding the function IComponent::isStructuredStorage (p. 10) to return true.

See IStructuredStorage (p. 711) for more information on structured storage.

Public Functions

Constructors

The constructors and destructors for this class are called by the framework. You do not need to manage storage objects directly.

IFlatFileStorage

IFlatFileStorage(IComponent&);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	N	N

This default constructor is called by the framework to create a storage object for components that use flat file storage. You do not need to explicitly instantiate storage objects.

IFlatFileStorage

~IFlatFileStorage

```
virtual  
~IFlatFileStorage();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

File I/O

Use these members to save or load a model to or from a file.

loadFile Loads a model from an ordinary flat file. This is a concrete implementation of IDocumentStorage::loadFile (p. 231).

```
virtual void  
loadFile( const IString& name );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IString& The name of the file to be loaded.

Exceptions	
IAccessError	Unexpected read error. Recoverable. This function also passes through many other stream-in and meta type system exceptions, for example, "invalid stream", "unsupported version".

saveFile Saves a model to an ordinary flat file. This is a concrete implementation of IDocumentStorage::saveFile (p. 232).

```
virtual void  
saveFile( const IString& name,  
          Boolean sameAsLoad,  
          Boolean remember );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

IString& The name of the file to be saved.

Boolean Indicates whether the name being used is the same as that used during loading.

Boolean Indicates that the file name should be retained.

Exceptions	
IAccessError	Unexpected write error. Recoverable.

IFlatFileStorage

Inherited Public Functions

IDocumentStorage		
component	fileName	loadFile

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

The constructors and destructors for this class are called by the framework. You do not need to manage storage objects directly.

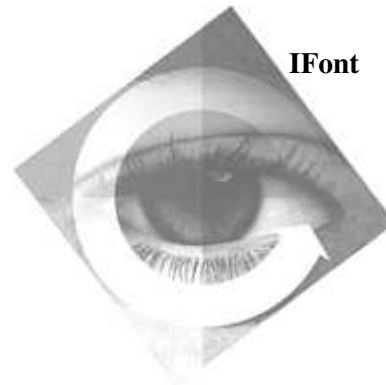
IFlatFileStorage

```
IFlatFileStorage( const IFlatFileStorage& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFont

IFont

Derivation

IBase
IVBase
IFont

Inherited By

None.

Header File

ifont.hpp

Members

Member	Page	Member	Page
Constructor	255	minTextWidth	261
avgCharWidth	259	name	263
avgLowercase	259	operator =	256
avgUppercase	259	pointSize	263
beginUsingFont	257	pointSizeAt	257
charWidth	259	setAllEmphasis	264
endUsingFont	257	setBold	264
externalLeading	259	setCharHeight	266
faceNameAt	257	setCharSize	266
fattr	254	setCharWidth	266
fontmetrics	255	setDirection	264
internalLeading	260	setFontAngle	266
isBitmap	263	setFontShear	266
isBitmapOnly	257	setItalic	265
isBold	263	setName	265
isFixed	263	setOutline	265
isItalic	263	setPointSize	265
isNonPropOnly	258	setStrikeout	265
isOutline	263	setUnderscore	265
isStrikeout	264	setWindowFont	267
isUnderscore	264	subscriptOffset	261
isVectorOnly	258	subscriptSize	261
logfont	255	superscriptOffset	262
maxAscender	260	superscriptSize	262
maxCharHeight	260	textLines	262
maxDescender	260	textWidth	262
maxLowercaseAscender	260	useBitmapOnly	258
maxLowercaseDescender	261	useNonPropOnly	258
maxSize	261	useVectorOnly	258
maxUppercaseSize	261	~IFont	257

The IFont class manages the use of fonts. Use this class to select a font through the IFont functions. You can also use the font dialog to do the following:

- Get font information
- Set the font when drawing text

IFont

IFont attempts to match the requested font. If it cannot find an exact match, IFont uses the nearest match.

Note: An IFont object represents a particular font that is available on the system. It does not represent the following:

- A font actually being used by a control
- The currently selected font for a presentation space

IWindow::setFont (Vol. II) changes the font used to draw text in a control.

beginUsingFont (p. 257) causes all subsequent text drawn to the specified presentation space to be in the font represented by the IFont object. You can use beginUsingFont in conjunction with an IPaintHandler (Vol. II) object.



Some of the set and use functions (such as setCharHeight and useBitmapOnly) accept an optional parameter of type IPresSpaceHandle (Vol. II). If you are using IFont to represent printer fonts, you must provide this parameter. If you do not specify a presentation space, IFont uses the presentation space of the desktop.



Some of the set and use functions, such as setCharHeight and useBitmapOnly, accept an optional parameter of type IPresSpaceHandle (Vol. II). Because the AIX version of IFont does *not* support printer fonts, there is no need to supply the optional IPresSpaceHandle parameter.

The members in this class that refer to vector fonts use scalable fonts in the X-Motif environment.



If you are porting an OS/2 Presentation Manager program to Motif, be aware that the font names vary between these systems. The only fonts that you can be reasonably sure of finding on both Motif and Presentation Manager are Courier and Helvetica. On either system, if you try to construct an IFont object for a nonexistent font, you will get Courier as the default font.

If you are porting an OS/2 Presentation Manager program to Windows, be aware that the font names vary between these systems. The list of system provided fonts are different.

Public Functions

Accessing Operating System Structures Related to Font

Use these members to access operating system specific font information.

fattrs Returns a pointer to the font's OS/2 Presentation Manager FATTRS structure.

IFont

	const struct _FATTRS* fattrs() const;	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
fontmetrics	Returns a pointer to the font's OS/2 Presentation Manager FONTMETRICS structure.			
	const struct _FONTMETRICS* fontmetrics() const;	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
logfont	Returns a pointer to the font's Windows LOGFONT structure.			
	const struct tagLOGFONTW const struct tagLOGFONTA* logfont() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N

Constructors

Use these members to construct, copy, assign and destruct objects of this class.

IFont

1	IFont(const IFont& fntCopy);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	--------------------------------	-----------------	----------------	-------------------

You can construct objects of this class from another IFont object. The parameter for this constructor is the following:

fntCopy The font you want to copy.

2	IFont(const IWindow* window = 0);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	-------------------------------------	-----------------	----------------	-------------------

You can construct objects of this class using a window's font. This constructs a font from the current font being used in the window. The parameter for this constructor is the following:

window The window to query for the font to use.

If you do not specify a window, the system default font is used.

PM In the OS/2 operating system, if the IWindow* is a multiline edit field (MLE), Presentation Manager returns the desktop presentation space instead of the MLE presentation space. In doing so, a different font is displayed.

IFont



In the AIX release, if you do not specify a window, the default font of the current display is used.

3	<pre>IFont(const char* faceName, unsigned long pointSize = 0, Boolean useFixedFont = false, Boolean useVectorFont = false, const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle ());</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

You can construct objects of this class by specifying the name and size of the font desired, along with additional parameters. The parameters for this constructor are the following:

faceName The face name of the font you want.

pointSize The specific point size you want for the font.

useFixedFont
 Specify whether you want the font fixed or proportional. The default is proportional.

useVectorFont
 Specify whether you want a vector or bitmap font. The default is a bitmap font.

presSpaceHandle
 The presentation space handle.

4	<pre>IFont(const IPresSpaceHandle& presSpaceHandle);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

You can construct objects of this class using a presentation space. This constructs a font corresponding to the font used in the specified presentation space. The parameter for this constructor is the following:

presSpaceHandle
 The presentation space handle.

operator = Assigns the value of one font object to another.

<pre>IFont& operator =(const IFont& font);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

~IFont

```
virtual
~IFont();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Cursor-Related Functions

Use these members with a cursor to retrieve font information.

faceNameAt Queries a font face name for the current position of the specified cursor.

```
static IString
faceNameAt( const FaceNameCursor& faceNameCursor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

pointSizeAt Queries the point size of a font for the current position of the specified cursor.

```
static long
pointSizeAt( const PointSizeCursor& pointSizeCursor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Drawing Functions

Use these members to draw text using a font.

beginUsingFont Sets the presentation space to use the font.

```
virtual IFont&
beginUsingFont( const IPresSpaceHandle& presSpaceHandle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Exceptions	
IAccessError	The operating system is unable to create or set the font.

endUsingFont Restores the presentation space to the default font.

```
virtual IFont&
endUsingFont( const IPresSpaceHandle& presSpaceHandle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Font Types

Use these members to query and set the types of fonts used by the IFont object.

isBitmapOnly If the IFont object uses only bitmap fonts, true is returned.

IFont

Boolean
isBitmapOnly() const;

Win
Y

PM
Y

Motif
Y

isNonPropOnly If the IFont object uses only nonproportional fonts, true is returned.

Boolean
isNonPropOnly() const;

Win
Y

PM
Y

Motif
Y

isVectorOnly If the IFont object uses only vector fonts, true is returned.

Boolean
isVectorOnly() const;

Win
Y

PM
Y

Motif
Y

useBitmapOnly Sets whether the IFont object uses only bitmap fonts.

virtual IFont&
useBitmapOnly(
Boolean bitmapOnly = true,
const IPresSpaceHandle& presSpaceHandle =
IPresSpaceHandle ());

Win
Y

PM
Y

Motif
Y

useNonPropOnly Sets whether the IFont object uses only nonproportional fonts.

virtual IFont&
useNonPropOnly(
Boolean nonProportionalOnly = true,
const IPresSpaceHandle& presSpaceHandle =
IPresSpaceHandle ());

Win
Y

PM
Y

Motif
Y

useVectorOnly Sets whether the IFont object uses only vector fonts.

virtual IFont&
useVectorOnly(
Boolean vectorOnly = true,
const IPresSpaceHandle& presSpaceHandle =
IPresSpaceHandle ());

Win
Y

PM
Y

Motif
Y

Geometry Accessors

Use these members to query information about the size of the font.

avgCharWidth

Returns the average character's width.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
avgCharWidth() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

avgLowercase

Returns the nominal height above the baseline for lowercase characters.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
avgLowercase() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

avgUppercase

Returns the height of the Em square. This corresponds to the point size.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
avgUppercase() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

charWidth

Returns the width of a specific single-byte character.

c A character. You can only specify SBCS characters.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
charWidth(char <i>c</i>) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

PM This member function is not DBCS-enabled.

Motif This member function is not MBCS-enabled.

externalLeading

Returns the amount of white space that appears between adjacent rows of text for this font.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
externalLeading() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

IFont

internalLeading

Returns the approximate position of the top of a row of characters. You can use this value to position the first line of a block of text by doing both of the following:

1. Subtracting the internalLeading value from the value returned by maxAscender (p. 260).
2. Positioning the baseline of the first line of text below the item (such as text or graphics) that is above the text. Use the value you obtained in step 1 to do so.

Note: There is no guarantee that the positioned characters will not write over the item that is above them. Test to see if anything is overwritten and allocate additional space, if necessary.

```
unsigned long  
    internalLeading() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

maxAscender

Returns the height of the largest ascender above the baseline.

```
unsigned long  
    maxAscender() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

maxCharHeight

Returns the height portion of IFont::maxSize (p. 261).

```
unsigned long  
    maxCharHeight() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

maxDescender

Returns the height of the largest descender below the baseline.

```
unsigned long  
    maxDescender() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

maxLowercaseAscender

Returns the height of the largest lowercase ascender.

```
unsigned long  
    maxLowercaseAscender() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>Y</i>

IFont

maxLowercaseDescender

Returns the height of the largest lowercase descender.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
maxLowercaseDescender() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

maxSize

Returns the maximum width and height.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
maxSize() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

maxUppercaseSize

Returns the maximum size for an uppercase character.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
maxUppercaseSize() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

minTextWidth

Returns the width of the widest word.

line Pointer to character string. You can only specify an SBCS string.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minTextWidth(const char* line) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

PM This member function is not DBCS-enabled.

Motif This member function is not MBCS-enabled.

subscriptOffset

Returns the recommended size of the baseline X-Y offset for subscripts.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
subscriptOffset() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

subscriptSize

Returns the recommended size for subscripts.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
subscriptSize() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

IFont

superscriptOffset

Returns the recommended size of the baseline X-Y offset for superscripts.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
superscriptOffset() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

superscriptSize

Returns the recommended size for superscripts.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
superscriptSize() const;	<i>N</i>	<i>Y</i>	<i>Y</i>

textLines

Returns the number of lines required to fit the specified text using the specified maximum line width.

text Pointer to character string. You can only specify an SBCS string.

lineWidth The maximum line width.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
textLines(const char* text, unsigned long lineWidth) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

PM This member function is not DBCS-enabled.

Motif This member function is not MBCS-enabled.

textWidth

Returns the width of the specified string.

1	unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	textWidth(const char* text, IBidiSettings& settings) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Returns the width of the specified string using the specified bidirectional attributes.

This member function is not DBCS-enabled.

text Pointer to character string. You can only specify an SBCS string.

2	unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	textWidth(const char* text) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Returns the width of the specified string. **This member function is not DBCS-enabled.**

IFont

text Pointer to character string. You can only specify an SBCS string.

PM This member function is not DBCS-enabled.

Motif This member function is not MBCS-enabled.

Getting Font Attributes

Use these members to query the attributes of the current font.

isBitmap If the IFont object uses a bitmap font, true is returned. If the IFont object uses a vector font, false is returned.

Boolean	Win	PM	Motif
isBitmap() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isFixed If the IFont object uses a fixed-size (that is, nonproportional) font, true is returned.

Boolean	Win	PM	Motif
isFixed() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

name Returns the face name of the font.

IString	Win	PM	Motif
name() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

pointSize Returns the point size of the font.

unsigned long	Win	PM	Motif
pointSize() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Getting Font Style

Use these members to query the appearance of a font.

isBold If the IFont object uses a font that is bold, true is returned.

Boolean	Win	PM	Motif
isBold() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isItalic If the IFont object uses an italicized font, true is returned.

Boolean	Win	PM	Motif
isItalic() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isOutline If the IFont object uses a font that appears hollow, true is returned.

IFont

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isOutline() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

isStrikeout If the IFont object uses a font with a line drawn through the characters, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isStrikeout() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

isUnderscore If the IFont object uses a font with a line drawn under the characters, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isUnderscore() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

Setting Font Direction

Use these members to set the direction that the font is drawn in.

setDirection Sets the direction to draw the font in.

direction Use the enumeration `Direction` (p. 268) to specify the direction.

virtual IFont&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDirection(<code>Direction direction</code>);	<i>Y</i>	<i>Y</i>	<i>I</i>



In the AIX release, the system locale information determines the font direction.

Setting Font Style

Use these members to change the appearance of a font.

setAllEmphasis

If you specify true, all of the styles are set on. If you specify false, all of the styles are set off.

virtual IFont&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setAllEmphasis(<code>Boolean turnOn = true</code>);	<i>Y</i>	<i>Y</i>	<i>Y</i>

setBold Changes to bold font.

virtual IFont&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setBold(<code>Boolean bold = true</code>);	<i>Y</i>	<i>Y</i>	<i>Y</i>

IFont

setItalic Changes to italic font.

```
virtual IFont&
    setItalic( Boolean italics = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setOutline Causes the font to appear hollow.

```
virtual IFont&
    setOutline( Boolean outline = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

setStrikeout Draws a line through the characters.

```
virtual IFont&
    setStrikeout( Boolean strikeout = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

setUnderscore

Draws a line under the characters.

```
virtual IFont&
    setUnderscore( Boolean underscore = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

Setting General Font Attributes

Use these members to set the attributes of any font being managed by the IFont class.

setName Sets the font's face name.

```
virtual IFont&
    setName(
        const char* name,
        const IPresSpaceHandle& presSpaceHandle =
            IPresSpaceHandle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setPointSize Sets the font's point size.

```
virtual IFont&
    setPointSize(
        unsigned long size,
        const IPresSpaceHandle& presSpaceHandle =
            IPresSpaceHandle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Setting Vector Font Attributes

Use these members to set the attributes of vector fonts. They have no effect when applied to a bitmap font.

IFont

setCharHeight

Sets the height of the characters of a vector font. This function has no effect when applied to a bitmap font.

virtual IFont& setCharHeight(unsigned long height, const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle ());	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

setCharSize

Sets the size of the characters of a vector font. This function has no effect when applied to a bitmap font.

virtual IFont& setCharSize(const ISize& size, const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle ());	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

setCharWidth

Sets the width of the characters of a vector font. This function has no effect when applied to a bitmap font.

virtual IFont& setCharWidth(unsigned long width, const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle ());	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

setFontAngle

Sets the angle to draw the vector font with. This function has no effect when applied to a bitmap font.

virtual IFont& setFontAngle(const IPoint& point, const IPresSpaceHandle& presSpaceHandle = IPresSpaceHandle ());	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

setFontShear

Sets the amount of shear to apply to the vector font. This function has no effect when applied to a bitmap font.

IFont

```
virtual IFont&
setFontShear(
    const IPoint& point,
    const IPresSpaceHandle& presSpaceHandle =
        IPresSpaceHandle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

Setting Window Font

Use these members to apply the current font to an IWindow

setWindowFont

Sets the font for the specified IWindow (Vol. II) to the current font. If the font is successfully changed, true is returned.

```
virtual Boolean
setWindowFont( IWindow* window ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Exceptions	
IAccessError	The operating system is unable to set the font for the window.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IFont contains the following nested classes:

IFont::PointSizeCursor (see page 272)

IFont

IFont::FaceNameCursor (see page 269)

Direction

```
Direction {  
    defaultDir,          leftToRight,          topToBottom,  
    rightToLeft,         bottomToTop,          leftRight = leftToRight,  
    topBottom = topToBottom, rightLeft = rightToLeft, bottomTop = bottomToTop  
};
```

Use these enumerators to specify the direction in which a font is drawn. You can use these enumerators as input to `setDirection` (p. 264).

defaultDir

Draws the font in the default direction.

leftRight

Draws the font from left to right.

topBottom

Draws the font from top to bottom.

rightLeft

Draws the font from right to left.

bottomTop

Draws the font from bottom to top.

Note: The following enumerators are obsolete; use their replacements instead:

Obsolete	Replacement
<code>bottomTop</code>	<code>bottomToTop</code>
<code>leftRight</code>	<code>leftToRight</code>
<code>rightLeft</code>	<code>rightToLeft</code>
<code>topBottom</code>	<code>topToBottom</code>



IFont::FaceNameCursor

Derivation IBase
IVBase
IFont::FaceNameCursor

Inherited By None.

Header File ifont.hpp

Members	Member	Page	Member	Page
	Constructor	269	setToLast	270
	FaceNameCursor	269	setToNext	270
	invalidate	270	setToPrevious	270
	isValid	270	~FaceNameCursor	270
	setToFirst	270		

The IFont::FaceNameCursor nested class iterates over the public and private fonts available for a presentation space.

Public Functions

Constructors

You can construct and destruct objects of this class.

FaceNameCursor

Constructs a IFont::FaceNameCursor using a FontType enumerator and a presentation space handle. Both parameters are optional. The FontType enumerator specifies which type you want to cursor through. If you do not specify a presentation space, the desktop presentation space is used.

fontType Type of font to cursor through.

presSpaceHandle

Presentation space handle used to query the available fonts. Optional.

```
FaceNameCursor(
    FontType fontType = both,
    const IPresSpaceHandle& presSpaceHandle =
        IPresSpaceHandle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

IFont::FaceNameCursor

~FaceNameCursor

```
virtual  
    ~FaceNameCursor();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Overrides

Use these members to iterate through the available font face names.

invalidate Marks the cursor as not valid.

```
virtual void  
    invalidate();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

isValid Returns true if the cursor is valid.

```
virtual Boolean  
    isValid() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setToFirst Advances the cursor position to the first face name.

```
virtual Boolean  
    setToFirst();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setToLast Advances the cursor position to the last face name.

```
virtual Boolean  
    setToLast();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setToNext Advances the cursor position to the next face name.

```
virtual Boolean  
    setToNext();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

setToPrevious

Advances the cursor position to the previous face name.

```
virtual Boolean  
    setToPrevious();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

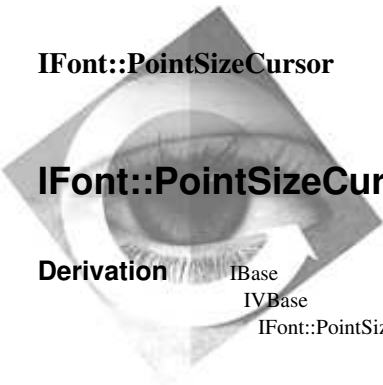
Inherited Protected Data

IBase		
recoverable	unrecoverable	

FontType

```
FontType {
    bitmap,
    vector,
    both
};
```

Use this enumeration to specify which types of fonts (bitmap, vector, or both) you want to iterate through.



IFont::PointSizeCursor

IFont::PointSizeCursor



Inherited By None.

Header File ifont.hpp

Members	Member	Page	Member	Page
	Constructor	272	setToLast	273
	invalidate	273	setToNext	273
	isValid	273	setToPrevious	273
	PointSizeCursor	272	~PointSizeCursor	273
	setToFirst	273		

The IFont::PointSizeCursor nested class iterates over the point sizes available for a font.

Public Functions

Constructors

You can construct and destruct objects of this class.

PointSizeCursor

Constructs an IFont::PointSizeCursor from the font face name and a presentation space handle. The presentation space handle is optional. If you do not specify it, the desktop presentation space is used.

facename Font face name.

presSpaceHandle
Presentation space handle used to query the point sizes. Optional.

```
PointSizeCursor(
    const char* facename,
    const IPresSpaceHandle& presSpaceHandle =
        IPresSpaceHandle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

IFont::PointSizeCursor

~PointSizeCursor

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~PointSizeCursor();	Y	Y	Y

Overrides

Use these members to iterate through the available point sizes for a given font.

invalidate Marks the cursor as not valid.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
invalidate();	Y	Y	Y

isValid Returns true if the cursor is valid.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isValid() const;	Y	Y	Y

setToFirst Advances the cursor position to the first point size.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setToFirst();	Y	Y	Y

setToLast Advances the cursor position to the last point size.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setToLast();	Y	Y	Y

setToNext Advances the cursor position to the next point size.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setToNext();	Y	Y	Y

setToPrevious

Advances the cursor position to the previous point size.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setToPrevious();	Y	Y	Y

IFont::PointSizeCursor

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IG3PointArc

Derivation

```

IBase
IVBase
IGraphic
IG3PointArc
  
```

Inherited By None.

Header File igarc.hpp

Members	Member	Page	Member	Page
	Constructor	276	operator ==	276
	drawOn	278	setEndingPoint	277
	endingPoint	277	setIntermediatePoint	277
	intermediatePoint	277	setStartingPoint	277
	operator !=	275	startingPoint	277
	operator =	276	~IG3PointArc	276

The IG3PointArc class is a graphic object class that allows you to create two-dimensional arcs by specifying three points the arc passes through. Unless you apply a transform to objects of this class, the three points will specify an arc of a circle.

When you draw an IG3PointArc, the following graphic bundle attributes affect its appearance:

- Pen color
- Mix mode
- Pen width
- Pen type
- Pen-ending style

Public Functions

Comparisons

Use these members to compare two IG3PointArc objects.

operator != Returns true if the arcs are not identical (includes the graphic bundle attributes and the transform matrix).

IG3PointArc

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IG3PointArc& arc) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the arcs are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IG3PointArc& arc) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of the this class.

IG3PointArc

1	IG3PointArc(const IPoint& starting, const IPoint& intermediate, const IPoint& ending);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct a IG3PointArc object from three points. The first point defines the starting location of the arc, the second point defines an intermediate point along the arc, and the last point defines the end point of the arc.

starting First point of the arc.
intermediate
Point on the arc between the starting and ending points.
ending End point of the arc.

2	IG3PointArc(const IG3PointArc& arc);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct a IG3PointArc object from another IG3PointArc object.

arc Reference to another IG3PointArc object.

operator = Use this function to assign a IG3PointArc object to another IG3PointArc object.

IG3PointArc&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IG3PointArc& arc);	<i>Y</i>	<i>Y</i>	<i>N</i>

~IG3PointArc

IG3PointArc

```
virtual  
~IG3PointArc();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Data Access

Use these members to set and query the points that the arc passes through.

endingPoint Returns the ending point of the arc.

```
virtual IPoint  
endingPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

intermediatePoint

Returns the intermediate point of the arc.

```
virtual IPoint  
intermediatePoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setEndingPoint

Sets the ending point of the arc.

```
virtual IG3PointArc&  
setEndingPoint( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setIntermediatePoint

Sets the intermediate point of the arc.

```
virtual IG3PointArc&  
setIntermediatePoint( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setStartingPoint

Sets the starting point of the arc.

```
virtual IG3PointArc&  
setStartingPoint( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

startingPoint Returns the starting point of the arc.

```
virtual IPoint  
startingPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IG3PointArc

Drawing

Use these members to render the graphic object.

drawOn Draws the arc on the device associated with the graphic context.

```
virtual IG3PointArc&
drawOn( IGraphicContext& graphicContext );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File igarc.hpp

Members				
Member	Page	Member	Page	
Constructor	281	setDirection	282	
direction	282	setEnclosingRect	283	
drawOn	282	setStartAngle	283	
enclosingRect	282	setSweepAngle	283	
operator !=	280	startAngle	283	
operator =	281	sweepAngle	283	
operator ==	281	~IGarc	282	

The IGarc class is a graphic object class that allows you to create two-dimensional arcs by specifying a bounding rectangle, a start angle, and a sweep angle. The rectangle you specify in the constructor is the enclosing rectangle of an ellipse. The start angle and sweep angle specify an arc section of this ellipse.

When you draw an IGarc, the following graphic bundle attributes affect its appearance:

- Pen color
- Mix mode
- Pen width
- Pen type
- Pen-ending style

Public Functions

Comparisons

Use these members to compare two IGarc objects.

operator != Returns true if the arcs are not identical (includes the graphic bundle attributes and the transform matrix).

IGArc

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGArc& arc) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the arcs are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGArc& arc) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGArc

1	IGArc(const IRectangle& rectangle, double startAngle, double sweepAngle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct a IGArc object from a IRectangle object and a start and sweep angle. The rectangle specifies an enclosing rectangle of an ellipse. The start and sweep angles define an arc section of the ellipse.

rectangle Defines the enclosing rectangle of an ellipse.

startAngle Defines the starting angle of an arc section of the ellipse defined by the IRectangle object.

sweepAngle Defines the sweep angle of an arc section of the ellipse defined by the IRectangle object.

2	IGArc(const IGArc& arc);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct a IGArc object from another IGArc object.

arc A reference to a IGArc object.

operator = Use this function to assign a IGArc object to another IGArc object.

IGArc&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IGArc& arc);	<i>Y</i>	<i>Y</i>	<i>N</i>

IGArc

~IGArc

<pre>virtual ~IGArc();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Direction

Use these members to set and query the direction that the arc is drawn in. The direction controls whether the sweep angle of the arc is drawn in a clockwise or counterclockwise direction.

direction Returns the direction the arc is drawn. The default direction is counterclockwise.

<pre>virtual Direction direction() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setDirection Sets the direction the arc is drawn.

<pre>virtual IGArc& setDirection(Direction direction = counterclockwise);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Drawing

Use these members to render the graphic object.

drawOn Draws the arc on the device associated with the graphic context.

<pre>virtual IGArc& drawOn(IGraphicContext& graphicContext);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Enclosing Rectangle

Use these members to set and query the enclosing rectangle of an ellipse. The arc is a section of this ellipse as specified by the start and sweep angles. The rectangle specifies an enclosing rectangle of an ellipse. The start and sweep angles define an arc section of the ellipse.

enclosingRect

Returns the enclosing rectangle of the ellipse that the arc is a section of.

<pre>virtual IRectangle enclosingRect() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IGArc

setEnclosingRect

Sets the enclosing rectangle of an ellipse. The arc is a section of this ellipse.

```
virtual IGarc&
    setEnclosingRect( const IRectangle& rectangle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Start and Sweep Angles

Use these members to set and query the start and sweep angles of the arc. The start angle specifies the angle where the arc begins. The sweep angle continues from the start angle, defining an arc section of the ellipse.

setStartAngle

Sets the start angle, in degrees, of the arc ($0 \leq \text{angle} \leq 360$).

```
virtual IGarc&
    setStartAngle( double startAngle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setSweepAngle

Sets the sweep angle, in degrees, of the arc ($0 \leq \text{angle} \leq 360$).

```
virtual IGarc&
    setSweepAngle( double sweepAngle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

startAngle Returns the start angle in degrees.

```
virtual double
    startAngle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

sweepAngle Returns the sweep angle in degrees.

```
virtual double
    sweepAngle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId

IGArc

IGraphic		
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Direction `Direction {
 clockwise,
 counterclockwise
 };`

The direction allows you to control the direction in which the arc is drawn from the start angle.

IGBitmap



Derivation IBase
IVBase
IGraphic
IGBitmap

Inherited By None.

Header File igbitmap.hpp

Members	Member	Page	Member	Page
	Constructor	289	reflectHorizontally	292
	asPointerHandle	286	reflectVertically	292
	black	296	resetTransparentColor	288
	copy	286	rotateBy180	292
	destInvert	296	rotateBy270	292
	drawOn	291	rotateBy90	293
	halfTone	297	setPalette	287
	handle	286	setTransparentColor	288
	hasTransparentColor	288	setViewOption	294
	imageFormat	294	size	287
	invert	297	sizeTo	287
	loadBitmap	295	sourceAnd	298
	mergeCopy	297	sourceErase	298
	mergePaint	297	sourceInvert	298
	moveTo	287	sourcePaint	298
	normal	297	transparentColor	288
	notSourceErase	297	transposeXForY	293
	palette	287	tryToLoadBitmap	295
	patternCopy	297	viewOption	294
	patternInvert	298	white	298
	patternPaint	298	writeToFile	294
	position	287	~IGBitmap	290

The IGBitmap class creates, modifies, and draws bitmaps. IGBitmap objects can be created from existing bitmap handles, from bitmap resources, from a rectangular area of a graphic context, or directly from an image file.

Once a bitmap has been created, you can save the bitmap in any of the supported image file formats.

If the bitmap is a color bitmap, none of the graphic bundle attributes affects the appearance of the bitmap. If the bitmap is a monochrome (1 bit-per-plane) bitmap, then the following graphic bundle attributes affect its appearance.

IGBitmap

- Pen color
- Background color
- Mix mode
- Background mix mode

Public Functions

Accessors

Use these members to return the bitmap as an IBitmapHandle object or as an IPointerHandle object.

asPointerHandle

Converts the bitmap to a pointer and returns a IPointerHandle object. It is the caller's responsibility to delete the pointer.

Note: The bitmap transparent color is not used when creating the pointer.

virtual IPointerHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
asPointerHandle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

handle

Returns the handle of the bitmap.

virtual IBitmapHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Bitmap Copying

Use these members to make a copy of the bitmap.

copy

Creates a copy of a bitmap. The caller must delete the bitmap. This function is useful when you need to create a bitmap and "give it away," as in placing a bitmap on the clipboard.

static IBitmapHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
copy(const IBitmapHandle& bitmapHandle);	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to copy the bitmap. Refer to the exception text for specific error information.

IGBitmap

Bitmap Palette

Use these members to set and query the color palette the bitmap uses when drawing the bitmap. The bitmap's color palette will be deleted when the bitmap is deleted.

palette Returns the handle to the color palette used when drawing the bitmap.

unsigned long palette() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
-----------------------------------	-----------------	----------------	-------------------

setPalette Sets the handle of the color palette to use when drawing the bitmap. The bitmap's previous palette, if any, is returned. Setting the color palette to zero means the bitmap will not select and realize a color palette prior to drawing.

unsigned long setPalette(unsigned long palette);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Bitmap Positioning

Use these members to set and query the size and position of the bitmap. The bitmap position information is not used during drawing if you use drawOn member, which accepts position arguments.

moveTo Sets the position of where the bitmap is drawn.

virtual IGBitmap& moveTo(const IPoint& point);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

position Returns the position of where the bitmap is drawn.

virtual IPoint position() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
-------------------------------------	-----------------	----------------	-------------------

size Returns the size of the bitmap.

virtual ISize size() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--------------------------------	-----------------	----------------	-------------------

sizeTo Resizes the bitmap to the argument size.

virtual IGBitmap& sizeTo(const ISize& newSize);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IGBitmap

Exceptions	
IAccessError	An error occurred while attempting to resize the bitmap. Refer to the exception text for specific error information.

Bitmap Transparency

Use these members to control transparency when you draw a bitmap on a graphic context. Bits whose color is the same as the transparency color are not visible. The object under the bitmap is visible for these bits. Use transparency when you want to mask out a section of the bitmap with the color you set as the transparency color.

Note: These members do not modify the actual bitmap data.

hasTransparentColor

Returns true if a transparent color has been set; otherwise, false is returned.

virtual IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasTransparentColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetTransparentColor

Resets the bitmap so that no color is treated as transparent when drawing the bitmap.

virtual IGBitmap&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetTransparentColor();	<i>Y</i>	<i>Y</i>	<i>N</i>

setTransparentColor

Sets the color to be treated as transparent when drawing the bitmap.

virtual IGBitmap&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setTransparentColor(const IColor& aColor);	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the bitmap transparency color. Refer to the exception text for specific error information.

transparentColor

Returns the current transparent color.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
transparentColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

IGBitmap

1	IGBitmap(const IBitmapHandle& bitmapHandle);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

Use this function to construct an IGBitmap object from an IBitmapHandle. This is useful for draw-item events where the bitmap handle is supplied.

bitmapHandle Handle to a bitmap.

2	IGBitmap(unsigned long bitmapIdentifier, IBase::Boolean useColorData = false);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

Use this function to construct an IGBitmap object from a resource identifier. The bitmap is loaded from the resource file bound to the executable or from the user's resource dynamic link library.

bitmapIdentifier Resource identifier.
useColorData Set to false if you want the bitmap created using the system palette. Set to true if you want the bitmap created using the color palette contained within the bitmap resource.

3	IGBitmap(const IResourceId& resourceIdentifier, IBase::Boolean useColorData = false);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

Use this function to construct an IGBitmap object from a IResourceId object. This allows you to specify a resource identifier that exists in another resource library.

resourceIdentifier A IResourceId object.
useColorData Set to false if you want the bitmap created using the system palette. Set to true if you want the bitmap created using the color palette contained within the bitmap resource.

4	IGBitmap(const IString& imageFilename);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

Use this constructor to create an IGBitmap object from an image file name. The image format (p. 299) is determined by the file extension of the image file name.

imageFilename Name of the image file you want to load.

IGBitmap

5	<code>IGBitmap(const IString& imageFilename, ImageFormat imageFormat);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct an IGBitmap object from an image file name and an image file format. This constructor is identical to the previous constructor except that you explicitly state the image file format. This is useful when you want to load an image from a file that does not have a file extension that matches an extension for a given image format. Refer to Image Formats (p. 299) for the file extensions that are associated with an image format.

imageFilename Name of the image file you want to load.
imageFormat Type of image format.

6	<code>IGBitmap(const IGraphicContext& graphicContext, const IRectangle& windowRectangle);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct an IGBitmap object from a graphic context and a rectangle specifying an area of the graphic context in device space.

graphicContext An IGraphicContext object.
windowRectangle The area of the graphic context you want to create a bitmap of.

7	<code>IGBitmap(const IGBitmap& bitmap, const IRectangle& rectangle, long rasterOperation = normal);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct an IGBitmap object from the rectangular portion of another IGBitmap object. You can optionally specify a raster operation that is used when the bitmap object is constructed.

bitmap A reference to a IGBitmap object.
rectangle Rectangle area of the bitmap to copy.
rasterOperation Raster operation used when copying the bitmap.

~IGBitmap

<code>virtual ~IGBitmap();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Drawing

Use these members to render a bitmap object on a device.

IGBitmap

drawOn Draws the bitmap on the device associated with the graphic context.

1

```
virtual IGBitmap&
    drawOn( IGraphicContext& graphicContext,
            long rasterOperation );
```

Win **PM** **Motif**
Y Y N

Draws the bitmap on the device associated with the graphic context. This function draws the bitmap at the bitmap's current position. You must specify a raster operation when using this function.

Exceptions	
IAccessError	An error occurred while attempting to draw the bitmap. Refer to the exception text for specific error information.

2

```
virtual IGBitmap&
    drawOn( IGraphicContext& graphicContext );
```

Win **PM** **Motif**
Y Y N

Draws the bitmap on the device associated with the graphic context. This function draws the bitmap at the bitmap's current position with a normal raster operation.

Exceptions	
IAccessError	An error occurred while attempting to draw the bitmap. Refer to the exception text for specific error information.

3

```
virtual IGBitmap&
    drawOn( IGraphicContext& graphicContext,
            const IPoint& targetBottomLeft,
            const IPoint& targetTopRight,
            const IPoint& sourceBottomLeft,
            const IPoint& sourceTopRight,
            long rasterOperation = normal,
            CompressMode compressMode = ignore );
```

Win **PM** **Motif**
Y Y N

Draws the bitmap on the device associated with the graphic context. You can shrink or stretch the bitmap when drawing the bitmap. The first pair of points allows you to specify a target rectangle where the bitmap is drawn; the second pair of points allows you to specify a rectangular area of the bitmap to draw into the target rectangle. You can optionally specify a compress mode and raster operation that will be used when you draw the bitmap.

Exceptions	
IAccessError	An error occurred while attempting to draw the bitmap. Refer to the exception text for specific error information.

IGBitmap

Modifying a Bitmap

Use these members to modify the bitmap. The rotate functions are different from IGraphic::rotateBy in that the change to the bitmap is permanent. IGraphic::rotateBy modifies a model transform that is set before drawing the bitmap and is removed after the bitmap has been drawn and does not affect the actual bitmap data. The rotations are counterclockwise.

reflectHorizontally

Reflects the bitmap from left to right.

<pre>virtual IGBitmap& reflectHorizontally();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Exceptions	
IAccessError	An error occurred while attempting to reflect the bitmap. Refer to the exception text for specific error information.

reflectVertically

Reflects the bitmap from top to bottom.

<pre>virtual IGBitmap& reflectVertically();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Exceptions	
IAccessError	An error occurred while attempting to reflect the bitmap. Refer to the exception text for specific error information.

rotateBy180 Rotates the bitmap by 180 degrees counterclockwise.

<pre>virtual IGBitmap& rotateBy180();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Exceptions	
IAccessError	An error occurred while attempting to rotate the bitmap. Refer to the exception text for specific error information.

rotateBy270 Rotates the bitmap by 270 degrees counterclockwise.

<pre>virtual IGBitmap& rotateBy270();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

IGBitmap

Exceptions	
IAccessError	An error occurred while attempting to rotate the bitmap. Refer to the exception text for specific error information.

rotateBy90 Rotates the bitmap by 90 degrees counterclockwise.

<code>virtual IGBitmap& rotateBy90();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to rotate the bitmap. Refer to the exception text for specific error information.

transposeXForY Transposes all X values for Y values.

<code>virtual IGBitmap& transposeXForY();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to transpose the bitmap. Refer to the exception text for specific error information.

View Option

Use these members to specify an algorithm to use when the image data is converted to a bitmap. The viewing options are as follows:

- raw

Indicates not to use any algorithm when converting the image data to a bitmap.

- errorDiffused

Indicates that an error diffusion algorithm is used when converting the image data to a bitmap. The actual image data is not affected, only what you see on the screen. Error diffusion takes longer than halftoning, but the bitmap is generally sharper, especially when viewing .gif files.

- halftoned

Indicates that a halftone algorithm is used when converting the image data to a bitmap. The actual image data is not affected, only what you see on the screen. Converting the image data using the halftoned algorithm is generally quicker than using the error diffusion algorithm.

IGBitmap

setViewOption

Sets the method used to convert the image data to a bitmap.

```
virtual IGBitmap&
    setViewOption( ViewOption viewOption = raw );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to change the view option. Refer to the exception text for specific error information.

viewOption Returns the method used to convert the image data to a bitmap.

```
virtual ViewOption
    viewOption() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Writing to a File

Use these members to save the bitmap in any of the supported image file formats and query the type of the image format.

imageFormat Returns the type of the bitmap image format.

```
virtual ImageFormat
    imageFormat() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

writeToFile Saves a bitmap to a file in any of the supported image file formats. If the specified file already exists, the file is overwritten.

```
virtual IGBitmap&
    writeToFile( const IString& imageFilename,
                ImageFormat imageFormat );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to write the image file. Check the path and filename.

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected

IGBitmap

IGraphic		
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Loading a Bitmap

These static members load a bitmap from a resource file.

loadBitmap Loads bitmaps from resource files. If you use the default for the size argument, the bitmap is not resized.

```
static unsigned long
loadBitmap( unsigned long bitmapIdentifier,
            const IModuleHandle& moduleHandle,
            const ISize& bitmapSize = ISize ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

tryToLoadBitmap

Loads bitmaps from resource files. If you use the default for the size argument, the bitmap is not resized. This function differs from loadBitmap in that it does not throw an exception if the bitmap cannot be loaded. This allows you to attempt to load a bitmap without having to catch an exception if the load fails.

```
static unsigned long
tryToLoadBitmap( unsigned long bitmapIdentifier,
                 const IModuleHandle& moduleHandle,
                 const ISize& bitmapSize = ISize ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGBitmap

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Public Data

Common Raster Operations

The following list defines 16 common raster operations for combining a source bitmap, destination bitmap (screen), and the current set pattern. You can specify these raster options when drawing a bitmap.

black	0
notSourceErase	\sim Source Destination
halftone	Pattern Source
invert	\sim Source
sourceErase	Source & \sim Destination
destInvert	\sim Destination
patternInvert	Pattern ^ Destination
sourceInvert	Source ^ Destination
sourceAnd	Source & Destination
mergePaint	\sim Source Destination
mergeCopy	Pattern & Source
normal	Source
sourcePaint	Source Destination
patternCopy	Pattern
patternPaint	Pattern \sim Source Destination
white	1

black The target bitmap is black.

```
static const long
black;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

destInvert The target bitmap is the inverse of the destination (screen) bitmap.

IGBitmap

	static const long destInvert;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
halftone	The target bitmap is determined by halftoning the source bitmap.			
	static const long halftone;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
invert	The target bitmap is determined by inverting the source bitmap.			
	static const long invert;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
mergeCopy	The target bitmap is determined by ANDing the current pattern set with the source bitmap.			
	static const long mergeCopy;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
mergePaint	The target bitmap is determined by ORing the destination (screen) bitmap with the inverse of the source bitmap.			
	static const long mergePaint;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
normal	The target bitmap is the source bitmap.			
	static const long normal;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
notSourceErase	The target bitmap is determined by ANDing the inverse of the destination (screen) bitmap with the inverse of the source bitmap.			
	static const long notSourceErase;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
patternCopy	The target bitmap is the current pattern set.			
	static const long patternCopy;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

IGBitmap

patternInvert The target bitmap is determined by XORing the current pattern set with the destination (screen) bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
patternInvert;	<i>Y</i>	<i>Y</i>	<i>N</i>

patternPaint The target bitmap is determined by ORing the current pattern set with the destination (screen) bitmap with the inverse of the source bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
patternPaint;	<i>Y</i>	<i>Y</i>	<i>N</i>

sourceAnd The target bitmap is determined by ANDing the destination (screen) bitmap with the source bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceAnd;	<i>Y</i>	<i>Y</i>	<i>N</i>

sourceErase The target bitmap is determined by ANDing the inverse of the destination (screen) bitmap with the source bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceErase;	<i>Y</i>	<i>Y</i>	<i>N</i>

sourceInvert The target bitmap is determined by XORing the destination (screen) bitmap with the source bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourceInvert;	<i>Y</i>	<i>Y</i>	<i>N</i>

sourcePaint The target bitmap is determined by ORing the destination (screen) bitmap with the source bitmap.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sourcePaint;	<i>Y</i>	<i>Y</i>	<i>N</i>

white The target bitmap is white.

static const long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
white;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	

CompressMode

```
CompressMode {
    bitAnd,      bitOr,      ignore,      or = bitOr,    and = bitAnd
};
```

The compress mode allows to specify how a bitmap will stretched if the bitmap is stretched when drawn. The compress mode values are:

bitAnd

Compresses the bitmap as necessary, using a logical AND operation on the eliminated rows and columns. This is useful for preserving the foreground when foreground bits are "0" and the background bits are "1".

bitOr

Compresses the bitmap as necessary, using a logical OR operation on the eliminated rows and columns. This is useful for preserving the foreground when foreground bits are "1" and the background bits are "0".

ignore

Compresses the bitmap as necessary, but ignores any eliminated rows or columns. This is useful for color bitmaps where the results of combining pels of different colors are unpredictable.

ImageFormat

```
ImageFormat {
    Bitmap, GIF,    PCX,    TIFF,    Targa,
    Amiga,  XBM,    PSEG
};
```

The IGBitmap class allows you to load and save bitmaps in several popular image file formats. There are two constructors that allow you to create a bitmap from an image file. You can use IGBitmap::writeToFile to save a bitmap in any of the supported image file formats. The image formats this class supports are:

OS/2 and Windows Bitmaps

The file extensions .BMP, .VGA, .BGA, .RLE, .DIB, .RL4, and .RL8 are recognized as OS/2 1.1, 1.2, 2.0 or Windows 3.0 bitmaps. The newer multimedia Windows bitmap format that allows 16 and 32 bits-per-plane is not supported. The files are written in OS/2 2.0/Windows 3.0 format.

IGBitmap

CompuServe Graphics Interchange Format

The .GIF file extension is recognized as a GIF file.

ZSoft PC Paintbrush Image File Format

The .PCX file extension is recognized as a Paintbrush file.

Microsoft/Aldus Tagged Image File Format

The .TIF and .TIFF file extensions are recognized as TIFF files.

Truevision Targa/Vista Bitmap

The file extensions .TGA, .VST, and .AFI are recognized as Targa/Vista files.

This class only supports 8 bit-per-plane and 24 bit-per-plane images.

Amiga IFF/ILBM Interleaved Bitmap Format

The file extensions .IFF and .LBM are recognized as interleaved bitmap files.

X Windows Bitmap

The .XBM file extension is recognized as a X Bitmap file. This class supports X10 and X11 1bpp bitmaps. Some .XBM files with text strings inside look to be sprites or icons and are not supported.

IBM Printer Page Segment

The following file extensions .PSE, .PSEG, .PSEG38PP and .PSEG3820 are recognized as PSEG files. PSEG files are used to include image data in BookMaster documents. PSEG files only contain 1 bit-per-plane, which is always ink on paper, ie: black on white.

ViewOption

```
ViewOption {  
    raw,  
    errorDiffused,  
    halftoned  
};
```

The viewing options allow you to specify an algorithm to use when the image data is converted to a bitmap. The viewing options are:

- raw

This indicates not to use any algorithm when converting the image data to a bitmap.

- errorDiffused

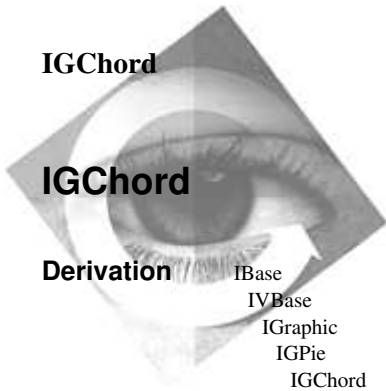
Indicates that an error diffusion algorithm is used when converting the image data to a bitmap. The actual image data is not affected, only what you see on the screen. Error diffusion takes longer than halftoning, but the bitmap is generally sharper, especially when viewing .gif files.

- halftoned

IGBitmap

Indicates that a halftoning algorithm is used when converting the image data to a bitmap. The actual image data is not affected, only what you see on the screen. Halftoning is generally quicker than error diffusion.

Note: Error diffusion and halftoning have little affect when used on OS/2 and Windows bitmaps.



Inherited By None.

Header File igpie.hpp

Members	Member	Page
	Constructor	303
	drawOn	303
	~IGChord	303

The IGChord class is a graphic object class that allows you to create a two-dimensional closed figure created from the chord of an ellipse. The IGChord can be filled, framed, or filled and framed. The rectangle you specify in the constructor is the enclosing rectangle of an ellipse. The start angle and sweep angle specify a chord section of this ellipse.

When you draw an IGChord, the following graphic bundle attributes affect its appearance:

- Draw operation
- Pen color
- Fill color
- Background color
- Mix mode
- Background mix mode
- Pen width
- Pen type
- Pen-ending style
- Pen-joining style
- Pen pattern
- Fill pattern
- Pattern origin

Public Functions

Constructors

You can construct, destruct, and copy objects of this class.

IGChord

1	IGChord(const IRectangle& rectangle, double startAngle, double sweepAngle);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Use this function to construct an IGChord object from an IRectangle object specifying the enclosing rectangle, a start angle, and a sweep angle. The sweep angle is drawn counterclockwise. The rectangle specifies an enclosing rectangle of an ellipse with the start and sweep angles defining a chord section of this ellipse.

<i>rectangle</i>	Enclosing rectangle of an ellipse.
<i>startAngle</i>	Start angle of the ellipse.
<i>sweepAngle</i>	Sweep angle specifying a chord-shaped section of the ellipse.

2	IGChord(const IGChord& chord);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	----------------------------------	-------------------------------	------------------------------	---------------------------------

Use this function to construct an IGChord object from another IGChord object.

<i>chord</i>	A reference to an IGChord object.
--------------	-----------------------------------

~IGChord

virtual ~IGChord();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
------------------------	-------------------------------	------------------------------	---------------------------------

Drawing

Use these members to render an IGChord object on a device.

drawOn Draws the IGChord object on the device associated with the graphic context.

virtual IGChord& drawOn(IGraphicContext& graphicContext);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

IGChord

Inherited Public Functions

IGPie		
drawOn	operator =	setStartAngle
enclosingRect	operator ==	setSweepAngle
operator !=	setEnclosingRect	startAngle

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IGEllipse

IGEllipse

Derivation IBase
IVBase
IGraphic
IGEllipse

Inherited By None.

Header File igellipse.hpp

Members	Member	Page	Member	Page
	Constructor	306	operator =	307
	drawOn	307	operator ==	306
	enclosingRect	307	setEnclosingRect	307
	operator !=	305	~IGEllipse	307

The IGEllipse class draws two-dimensional ellipses. An IGEllipse can be filled, framed, or filled and framed.

When you draw an IGEllipse, the following bundle attributes affect its appearance:

- Draw operation
- Pen color
- Fill color
- Background color
- Mix mode
- Background mix mode
- Pen width
- Pen type
- Pen pattern
- Fill pattern
- Pattern origin

Public Functions

Comparisons

Use these members to compare two IGEllipse objects.

operator != Returns true if the ellipses are not identical (includes the graphic bundle attributes and the transform matrix).

IGEllipse

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGEllipse& ellipse) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the ellipses are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGEllipse& ellipse) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of the IGEllipse class.

IGEllipse

1	IGEllipse(const IGEllipse& ellipse);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGEllipse object from another IGEllipse object.

ellipse A reference to an IGEllipse object.

2	IGEllipse(const IRectangle& rectangle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGEllipse object from an IRectangle object specifying the enclosing rectangle of the ellipse.

rectangle An IRectangle object specifying the enclosing rectangle of the ellipse.

3	IGEllipse(const IPoint& point, unsigned long radius);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGEllipse object from an IPoint object specifying the center of the ellipse and a radius indicating the length of the major and minor axes.

point Center point of the ellipse.

radius Radius of the ellipse.

4	IGEllipse(const IPoint& point, unsigned long xRadius, unsigned long yRadius);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

IGEllipse

Use this function to construct an IGEllipse object from an IPoint object specifying the center of the ellipse and two values indicating the length of the horizontal and vertical radii respectively.

point Center point of the ellipse.
xRadius Horizontal radius of the ellipse.
yRadius Vertical radius of the ellipse.

operator = Use this function to assign a IGEllipse object to another IGEllipse object.

IGEllipse& operator =(const IGEllipse& ellipse);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

~IGEllipse

virtual ~IGEllipse();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--------------------------	-------------------------------	------------------------------	---------------------------------

Drawing

Use these members to render an IGEllipse object on a device.

drawOn Draw the ellipse on the device associated with the graphic context.

virtual IGEllipse& drawOn(IGraphicContext& graphicContext);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

Enclosing Rectangle

Use these members to specify a rectangle that contains an ellipse.

enclosingRect

Returns the enclosing rectangle that the IGEllipse is contained within.

virtual IRectangle enclosingRect() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

setEnclosingRect

Sets the enclosing rectangle that the IGEllipse is contained within.

virtual IGEllipse& setEnclosingRect(const IRectangle& rectangle);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

IGEllipse

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IGLine



Derivation IBase
IVBase
IGraphic
IGLine

Inherited By None.

Header File igline.hpp

Members	Member	Page	Member	Page
	Constructor	310	operator ==	310
	centerPoint	311	setEndingPoint	311
	drawOn	311	setStartingPoint	311
	endingPoint	311	slope	311
	operator !=	309	startingPoint	311
	operator =	310	~IGLine	310

The IGLine class is a graphic object class that allows you to create two-dimensional line segments.

When you draw a IGLine, the following graphic bundle attributes affect its appearance:

- Pen color
- Mix mode
- Pen width
- Pen type
- Pen-ending style

Public Functions

Comparisons

Use these members to compare two IGLine objects.

operator != Returns true if the lines are not identical (includes the graphic bundle attributes and the transform matrix).

```
IBase::Boolean  
operator !=( const IGLine& line ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IGLine

operator == Returns true if the lines are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGLine& line) const;	Y	Y	N

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGLine

1	IGLine(const IPoint& startingPoint, const IPoint& endingPoint);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use this function to construct an IGLine object from two IPoint objects indicating the starting and ending points of the line.

<i>startingPoint</i>	Starting point of the line.
<i>endingPoint</i>	Ending point of the line.

2	IGLine(const IGLine& line);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use this function to construct a IGLine object from another IGLine object.

<i>line</i>	A reference to a IGLine object.
-------------	---------------------------------

operator = Use this function to assign a IGLine object to another IGLine object.

IGLine&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IGLine& line);	Y	Y	N

~IGLine

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IGLine();	Y	Y	N

Data Access

Use these members to query and set the data points of the line as well as to query the line's slope.

IGLine

centerPoint Returns the center point of the line segment.

```
virtual IPoint  
    centerPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

endingPoint Returns the ending point of the line segment.

```
virtual IPoint  
    endingPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setEndingPoint

Sets the ending point of the line segment.

```
virtual IGLine&  
    setEndingPoint( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setStartingPoint

Sets the starting point of the line segment.

```
virtual IGLine&  
    setStartingPoint( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

slope Returns the slope of the line segment (rise over run).

```
double  
    slope() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

startingPoint Returns the starting point of the line segment.

```
virtual IPoint  
    startingPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drawing

Use these members to render a IGLine object on a device.

drawOn Draws the line segment on the device associated with the graphic context.

```
virtual IGLine&  
    drawOn( IGraphicContext& graphicContext );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGLine

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IGList

Derivation

IBase
IVBase
IGraphic
IGList

Inherited By

None.

Header File

iglist.hpp

Members

Member	Page	Member	Page
Constructor	315	isIGList	317
addAsFirst	314	lastGraphic	317
addAsLast	314	numberOfGraphics	317
addAsNext	314	removeAll	318
addAsPrevious	314	removeAllWithId	318
addAtPosition	314	removeAt	318
bottomGraphicUnderPoint	316	removeAtPosition	318
boundingRectAt	316	removeFirst	318
drawOn	315	removeLast	318
firstGraphic	317	replaceAt	314
graphicAt	317	sort	318
graphicAtPosition	317	topGraphicUnderPoint	316
isEmpty	317	~IGList	315



The IGList class is an ordered collection of IGraphic objects. The IGraphic objects are arranged so that each IGList has a first and a last IGraphic object, each IGraphic object except the last has a next IGraphic object, and each IGraphic object but the first has a previous IGraphic object.

An IGList allows you to group simple IGraphic objects to compose a complex picture. At any time you can add additional IGraphic objects to the IGList or remove any IGraphic objects from it. You can also add the same IGraphic object to the list multiple times to replicate part of a picture.

Because IGList inherits from IGraphic, you can add an IGList to an IGList. You can also use any of the transform functions inherited from IGraphic on an IGList. Transforms applied to an IGList affect all IGraphic objects in the IGList. You can easily construct complex pictures and transform them as a single entity.

When you draw an IGList, it iterates through the graphic objects contained in the list. IGList recursively calls the drawOn function for all nested IGLists.

IGList

When you draw an IGList, the attributes you use to draw the graphic objects contained in the list are those of the graphic bundle applied to the IGList with the following exception. When a graphic object contains a graphic bundle that has the same attributes set, the graphic object's bundle attributes override the IGList's graphic bundle attributes.

Public Functions

Adding Graphic Objects

Use these members to add graphic objects to an IGList object.

addAsFirst Adds a graphic object as the first item in the list.

<pre>virtual IGList& addAsFirst(IGraphic& graphic);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsLast Adds a graphic object as the last item in the list.

<pre>virtual IGList& addAsLast(IGraphic& graphic);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsNext Adds a graphic object directly after the cursor location.

<pre>virtual IGList& addAsNext(const Cursor& cursor, IGraphic& graphic);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsPrevious

Adds a graphic object directly before the cursor location.

<pre>virtual IGList& addAsPrevious(const Cursor& cursor, IGraphic& graphic);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAtPosition

Adds a graphic object at the 0-based offset from the beginning of the list.

<pre>virtual IGList& addAtPosition(unsigned long position, IGraphic& graphic);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

replaceAt Replaces a graphic object at the cursor location.

IGList

```
virtual IGList&
    replaceAt( const Cursor& cursor,
               IGraphic& graphic );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, and copy objects of this class.

IGList

1 `IGList();`

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList object. The list is initially empty.

2 `IGList(IGraphic& graphic);`

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList object from an IGraphic object. The object is added to the list.

graphic A reference to an IGraphic object. This graphic is added to the list.

3 `IGList(const IGList& list);`

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList object from another IGList object. A copy of the graphic object references is created.

list A reference to an IGList object. All IGraphic objects in *list* are added to the new IGList.

~IGList

```
virtual
    ~IGList();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drawing

Use these members to render the graphic objects contained in an IGList on a device.

drawOn Iterates over the ordered collection calling each graphic object's drawOn function.

IGList

<pre>virtual IGList& drawOn(IGraphicContext& graphicContext);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Hit Testing

Use these members to determine if a point is contained within one of the graphic objects contained in the IGList.

Note: If you want to determine if more than one object is under a point, construct a Cursor object, which iterates through all objects that are under a specified point.

bottomGraphicUnderPoint

Returns the bottommost graphic object under the point specified. If a graphic object is not under the point specified, a null pointer is returned.

<pre>virtual IGraphic* bottomGraphicUnderPoint(const IPoint& point, IGraphicContext& graphicContext);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

topGraphicUnderPoint

Returns the topmost graphic object under the point specified. If a graphic object is not under the point specified, a null pointer is returned.

<pre>virtual IGraphic* topGraphicUnderPoint(const IPoint& point, IGraphicContext& graphicContext);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Querying the List

Use these members to query list information and to retrieve the graphic objects contained in the list.

boundingRectAt

Returns the smallest rectangle enclosing the graphic object at the cursor location. Use this function with a cursor constructed to locate objects under the mouse. The bounding rectangle returned takes into account cumulative transformations that may be set prior to drawing a graphic object in the list.

<pre>virtual IRectangle boundingRectAt(const Cursor& cursor) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IGList

firstGraphic Returns a pointer to the first graphic object in the list. If the list is empty, a null pointer is returned.

<code>virtual IGraphic*</code> <code>firstGraphic() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

graphicAt Returns a reference to the graphic object at the cursor location.

<code>virtual IGraphic&</code> <code>graphicAt(const Cursor& cursor) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

graphicAtPosition

Returns a reference to the graphic object at the given position in the list. Position 1 specifies the first graphic object.

<code>virtual IGraphic&</code> <code>graphicAtPosition(unsigned long position) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

isEmpty Returns true if the list is empty.

<code>virtual IBase::Boolean</code> <code>isEmpty() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

isIGList Returns true if the graphic object at the cursor location is an IGList.

<code>virtual IBase::Boolean</code> <code>isIGList(const Cursor& cursor) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

lastGraphic Returns a pointer to the last graphic object in the list. If the list is empty, a null pointer is returned.

<code>virtual IGraphic*</code> <code>lastGraphic() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

numberOfGraphics

Returns the count of the number of graphic objects in the list.

<code>virtual unsigned long</code> <code>numberOfGraphics() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGList

Removing Graphic Objects

Use these members to remove graphic objects from the IGList object.

removeAll Removes all graphic objects from the list.

<code>virtual IGList& removeAll();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeAllWithId

Removes all graphic objects that have the specified identifier.

<code>virtual IGList& removeAllWithId(unsigned long identifier);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeAt Removes the graphic object at the cursor location.

<code>virtual IGList& removeAt(const Cursor& cursor);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeAtPosition

Removes the graphic object at the 0-based offset from the beginning of the list.

<code>virtual IGList& removeAtPosition(unsigned long position);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeFirst Removes the first graphic object in the list.

<code>virtual IGList& removeFirst();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removeLast Removes the last graphic object in the list.

<code>virtual IGList& removeLast();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Reordering the List

Use these members to reorder the graphic objects in the IGList.

sort Reorders the graphics object contained in the graphic list. You need to write a C function with the following declaration:

IGList

```
long MyGraphicCompare( IGraphic& graphic1, IGraphic& graphic2 )
{
}
```

The function should return a long value:

Less than 0 The first object is less than the second object.
0 The first object is equal to the second object.
Greater than 0 The first object is greater than the second object.

The graphic objects are reordered in ascending order.

```
virtual IGList&
    sort(
        long ( * comparisonFunction) ( IGraphic * const & graphic1 ,
        IGraphic * const & graphic2 ) );
```

Win PM Motif
Y Y N

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGList

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IGList contains the following nested classes:

IGList::Cursor (see [page 321](#))



IGList::Cursor

Derivation IBase
 IVBase
 IGList::Cursor

Inherited By None.

Header File iglist.hpp

Members	Member	Page	Member	Page
	Constructor	321	setToLast	323
	Cursor	321	setToNext	323
	invalidate	322	setToPrevious	323
	isValid	323	~Cursor	322
	setToFirst	323		

The IGList::Cursor class iterates through graphic objects contained in an IGList. The Cursor class has three constructors that control how to iterate through the IGList. The class iterates through top-level objects only. If you nest an IGList inside an IGList, the cursor does not iterate through the graphic objects contained within the nested IGList.

The Cursor class iterates through all objects in an IGList, iterates through all objects with a specified identifier, or provides hit testing by iterating through all objects that are under a specified point. If you iterate by identifier, a match is determined by querying the object's identifier value that is set by the IGraphic::setId function.

If you want to iterate through nested IGLists, you can obtain a reference to the nested IGList (or any other graphic object in the list) by calling IGList::graphicAtPosition. You can then create another cursor for this list.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor Use objects of this class to cursor over graphic objects contained within an IGList.

IGList::Cursor

1	<code>Cursor(const IGList& list);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList::Cursor object from an IGList object. The cursor iterates through all graphic objects contained in the IGList.

list IGList to cursor through.

2	<code>Cursor(const IGList& list, unsigned long objectIdentifier);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList::Cursor object from an IGList object and an object identifier. The cursor iterates through graphic objects that have the specified object identifier.

list IGList to cursor through.
objectIdentifier Graphic objects identifier.

3	<code>Cursor(const IGList& list, IGraphicContext& graphicContext, const IPoint& hitPoint);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGList::Cursor object from an IGList object, an IGraphicContext object, and an IPoint object. The cursor iterates through graphic objects that intersect or contain the point specified.

list IGList to cursor through.
graphicContext Graphic context used to determine a graphic object's location.
hitPoint Position of the center of the hit aperture.

~Cursor

<code>virtual ~Cursor();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Cursor Positioning

Use these members to move the cursor through the list of graphic objects and to check the validity of the cursor.

invalidate Marks the cursor as invalid.

<code>virtual void invalidate();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGList::Cursor

isValid Returns true if the cursor is in a valid area.

```
virtual IBase::Boolean  
    isValid() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setToFirst Resets the cursor position to the first graphic object in the list.

```
virtual IBase::Boolean  
    setToFirst();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setToLast Resets the cursor position to the last graphic object in the list.

```
virtual IBase::Boolean  
    setToLast();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setToNext Advances the cursor position to the next graphic object in the list.

```
virtual IBase::Boolean  
    setToNext();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setToPrevious

Advances the cursor position to the previous graphic object in the list.

```
virtual IBase::Boolean  
    setToPrevious();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IGList::Cursor

IBase		
recoverable	unrecoverable	

IGPie

Derivation

IBase
IVBase
IGraphic
IGPie

Inherited By

IGChord

Header File

igpie.hpp

Members

Member	Page	Member	Page
Constructor	326	setEnclosingRect	327
drawOn	327	setStartAngle	328
enclosingRect	327	setSweepAngle	328
operator !=	326	startAngle	328
operator =	327	sweepAngle	328
operator ==	326	~IGPie	327



The IG Pie class is a graphic object class that allows you to create a two-dimensional pie slice of an ellipse. The IG Pie can be filled, framed, or filled and framed. The rectangle you specify in the constructor is the enclosing rectangle of an ellipse. The start angle and sweep angle specify a pie section of this ellipse.

When you draw an IG Pie, the following bundle attributes affect its appearance:

- Draw operation
- Pen color
- Fill color
- Background color
- Mix mode
- Background mix mode
- Pen width
- Pen type
- Pen-ending style
- Pen-joining style
- Pen pattern
- Fill pattern
- Pattern origin

IGPie

Public Functions

Comparisons

Use these members to compare two IGPie objects.

operator != Returns true if the pies are not identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGPie& pie) const;	Y	Y	N

operator == Returns true if the pies are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGPie& pie) const;	Y	Y	N

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGPie

1	IGPie(const IRectangle& rectangle, double startAngle, double sweepAngle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use this function to construct an IGPie object from an IRectangle object specifying the enclosing rectangle, a start angle, and a sweep angle. The sweep angle is drawn counterclockwise. The rectangle specifies an enclosing rectangle of an ellipse with the start and sweep angles defining a pie section of this ellipse.

<i>rectangle</i>	Enclosing rectangle of an ellipse.
<i>startAngle</i>	Start angle of the ellipse.
<i>sweepAngle</i>	Sweep angle specifying a pie-shaped section of the ellipse.

2	IGPie(const IGPie& pie);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use this function to construct an IGPie object from another IGPie object.

<i>pie</i>	A reference to an IGPie object.
------------	---------------------------------

IGPie

operator = Use this function to assign a IGPie object to another IGPie object.

IGPie& operator =(const IGPie& pie);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

~IGPie

virtual ~IGPie();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------------------	-----------------	----------------	-------------------

Drawing

Use these members to render an IGPie object on a device.

drawOn Draws the pie-shaped figure on the device associated with the graphic context.

virtual IGPie& drawOn(IGraphicContext& graphicContext);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Enclosing Rectangle

Use these members to specify a rectangle that contains an ellipse. The start and sweep angle define a pie-shaped section of this ellipse.

enclosingRect

Returns the enclosing rectangle of the ellipse that the pie is a section of.

virtual IRectangle enclosingRect() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

setEnclosingRect

Sets the enclosing rectangle of an ellipse. The pie is a section of this ellipse.

virtual IGPie& setEnclosingRect(const IRectangle& rectangle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Start and Sweep Angles

Use these members to define a pie-shaped section of an ellipse contained within the enclosing rectangle. The start and sweep angles are expressed in degrees.

IGPie

setStartAngle

Sets the start angle of the pie ($0 \leq \text{angle} \leq 360$).

<pre>virtual IGPie& setStartAngle(double startAngle);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

setSweepAngle

Sets the sweep angle of the pie ($0 \leq \text{angle} \leq 360$). The sweep angle is drawn in a counterclockwise direction.

<pre>virtual IGPie& setSweepAngle(double sweepAngle);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

startAngle Returns the start angle of the pie.

<pre>virtual double startAngle() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

sweepAngle Returns the sweep angle of the pie.

<pre>virtual double sweepAngle() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

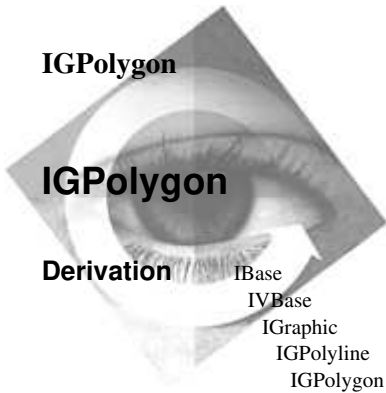
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File igpyline.hpp

Members		Member	Page	Member	Page
	Constructor		331	operator =	332
	drawOn		332	operator ==	331
	fillMode		332	setFillMode	332
	operator !=		331	~IGPolygon	332

The IGPolygon class is a graphic object class that allows you to create a two-dimensional closed figure from a series of line segments. The series of line segments is drawn starting from the first point and connecting all remaining points. If the first and last points are not the same, this class draws a line from the last point to the first point to close the figure.

You can optionally specify a fill mode to use when drawing the polygon. The default fill mode is alternate. Refer to the operating system's graphic programming reference for an explanation of these fill modes.

When you draw an IGPolygon, the following graphic bundle attributes affect its appearance:

- Draw operation
- Pen color
- Fill color
- Background color
- Mix mode
- Background mix mode
- Pen width
- Pen type
- Pen-ending style
- Pen-joining style
- Pen pattern

IGPolygon

- Fill pattern
- Pattern origin

Public Functions

Comparisons

Use these members to compare two IGPolygon objects.

operator != Returns true if the polygons are not identical (includes the graphic bundle attributes and the transform matrix).

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGPolygon& polygon) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the polygons are identical (includes the graphic bundle attributes and the transform matrix).

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGPolygon& polygon) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGPolygon

1	IGPolygon(const IPointArray& pointArray);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGPolygon object from an IPointArray (Vol. I) object. The polygon is constructed by drawing line segments starting from the first point connecting the remaining points. If the first and last points are not the same, a line is drawn from the first point to the last point to close the figure.

pointArray An array of data points.

2	IGPolygon(const IGPolygon& polygon);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGPolygon from another IGPolygon object.

polygon A reference to a IGPolygon object.

IGPolygon

operator = Use this function to assign an IGPolygon object to another IGPolygon object.

```
IGPolygon&
operator =( const IGPolygon& polygon );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IGPolygon

```
virtual
~IGPolygon();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drawing

Use these members to render an IGPolygon object on a device.

drawOn Draws the polygon on the device associated with the graphic context.

```
virtual IGPolygon&
drawOn( IGraphicContext& graphicContext );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Fill Mode

Use these members to query and set the fill mode that you use when drawing the polygon.

fillMode Returns the fill mode used when drawing the polygon.

```
virtual FillMode
fillMode() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setFillMode Sets the fill mode used when you draw the polygon.

```
virtual IGPolygon&
setFillMode( FillMode fillmode );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGPolyline		
addPoint	operator !=	pointArray
drawOn	operator =	removePoint
insertPoint	operator ==	reversePoints
numberOfPoints	point	setPoint

IGPolygon

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

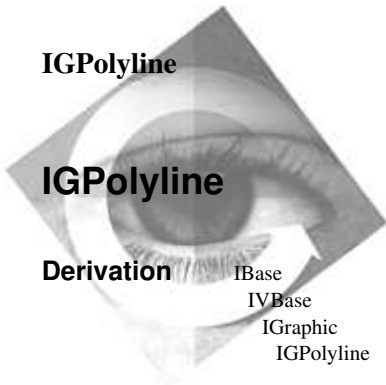
Inherited Protected Data

IBase		
recoverable	unrecoverable	

FillMode

```
FillMode {  
    alternate,  
    winding  
};
```

The fill mode allows you to control the method used to fill the interior of an IGPolygon.



Inherited By IGPolygon

Header File igpyline.hpp

Members		Member	Page	Member	Page
		Constructor	335	point	336
		addPoint	336	pointArray	336
		drawOn	337	removePoint	336
		insertPoint	336	reversePoints	336
		numberOfPoints	336	setPoint	337
		operator !=	334	setPoints	337
		operator =	335	~IGPolyline	335
		operator ==	335		

The IGPolyline class is a graphic object class that allows you to create a series of two-dimensional line segments. The series of line segments is drawn starting from the first point and connecting all remaining points.

When you draw an IGPolyline, the following graphic bundle attributes affect its appearance:

- Pen color
- Mix mode
- Pen width
- Pen type
- Pen-ending style

Public Functions

Comparisons

Use these members to compare two IGPolyline objects.

operator != Returns true if the polylines are not identical (includes the graphic bundle attributes and the transform matrix).

IGPolyline

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGPolyline& polyline) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the polylines are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGPolyline& polyline) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGPolyline

1	IGPolyline(const IPointArray& pointArray);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGPolyline object from an IPointArray (Vol. I) object. The polyline is constructed by drawing line segments starting from the first point and connecting the remaining points.

pointArray An array of data points.

2	IGPolyline(const IGPolyline& polyline);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct a IGPolyline object from another IGPolyline object.

polyline A reference to a IGPolyline object.

operator = Use this function to assign a IGPolyline object to another IGPolyline object.

IGPolyline&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IGPolyline& polyline);	<i>Y</i>	<i>Y</i>	<i>N</i>

~IGPolyline

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IGPolyline();	<i>Y</i>	<i>Y</i>	<i>N</i>

IGPolyline

Data Points

Use these members to add, modify, and remove the points that define the polyline.

addPoint Adds a point to the end of the polyline.

<code>virtual IGPolyline& addPoint(const IPoint& point);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

insertPoint Adds a point to the polyline at the location before the specified index. The index is 0-based.

<code>virtual IGPolyline& insertPoint(unsigned long index, const IPoint& point);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

numberOfPoints

Returns the number of points defining the polyline.

<code>virtual unsigned long numberOfPoints() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

point Returns a constant reference to the point at the specified index. The index is 0-based.

<code>const IPoint& point(unsigned long index) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

pointArray Returns a constant reference to the array of points defining the polyline.

<code>const IPointArray& pointArray() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

removePoint Removes a point for the polyline at the specified index. The index is 0-based.

<code>virtual IGPolyline& removePoint(unsigned long index);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

reversePoints

Reverses all points in the polyline. The first point becomes the last and so on.

<code>virtual IGPolyline& reversePoints();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGPolyline

setPoint Sets the point at the specified index to the argument point. The index is 0-based.

```
virtual IGPolyline&
    setPoint( unsigned long index,
              const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setPoints Sets the points used to define the polyline.

```
virtual IGPolyline&
    setPoints( const IPointArray& pointArray );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drawing

Use these members to render an IGPolyline object on a device.

drawOn Draws the polyline on the device associated with the graphic context.

```
virtual IGPolyline&
    drawOn( IGraphicContext& graphicContext );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IGPolyline

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IGrabHandles

Derivation IBase
IVBase
IGrabHandles

Inherited By None.

Header File igrabhdl.hpp

Members	Member	Page	Member	Page
	Constructor	339	getOutline	340
	changeTo	340	mouseDown	341
	draw	341	moveTo	340
	getLocation	340	~IGrabHandles	339

Objects of this class provide move/resize handles on an object being displayed in a window. The grabbable protocol must be implemented by (or on behalf of) the object being displayed. See IMGrabbable (p. 437) for more information.

Public Functions

Constructors

You can construct and destruct objects of this class.

IGrabHandles

```
IGrabHandles( IMGrabbable&,                               Win PM Motif  
              Boolean dragMove = false );                  Y   N   N
```

Begins displaying grab handles for the specified grabbable object. The grab handles start handling mouse events for the grabbed object's window at this point. If the second parameter, dragMove, is true, then move events are delegated to the grabbable object, which uses IMGrabbable::doDragDrop (p. 437) to initiate a drag-drop operation. If dragMove is false, the move events are handled by the grab handles, but only within the grabbable object's window.

~IGrabHandles

IGrabHandles

<pre>virtual ~IGrabHandles();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Location/Positioning

Use these members to obtain location information about, and set positions for, grab handles.

changeTo Move and/or resize the grab handles.

<pre>virtual void changeTo(const IRectangle& newRect);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

IRectangle&
The new location.

getLocation Returns the location value of the specified point, relative to the grab handles. The location is one of the following:

- kInside
- kOutside
- kTopLeft
- kTopMiddle
- kTopRight
- kMiddleLeft
- kMiddleRight,
- kBottomLeft
- kBottomMiddle
- kBottomRight.

<pre>virtual ELocation getLocation(const IPoint& position);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

IPoint& The point whose location is to be determined.

getOutline Returns the current bounding rectangle of the grab handles.

<pre>virtual IRectangle& getOutline() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

moveTo Moves the grab handles to the specified point.

IGrabHandles

```
virtual void  
    moveTo( const IPoint& newPoint );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IPoint& The new location for the grab handles.

Rendering/Control

Use these members to draw grab handles and give them responsibility for mouse events.

draw Draws the grab handles in the specified presentation space. If the second parameter is true, then a solid rectangle with the move and resize handles is drawn. If the second parameter is false, only a dashed outline rectangle is drawn.

```
virtual void  
    draw( IPresSpaceHandle& ps,  
          Boolean bWithHandles = true ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

IPresSpaceHandle&
 The presentation-space handle.
Boolean Indicates whether grab handles should be drawn.

mouseDown Informs the grab handles that the mouse is in the down position. This function should be called immediately after constructing grab handles if they were constructed as the result of a mouse down event. This function puts the grab handles into the correct initial state for them to assume responsibility for handling mouse events.

```
virtual void  
    mouseDown( const IPoint& position );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IPoint& The initial position of the mouse pointer.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

IGrabHandles

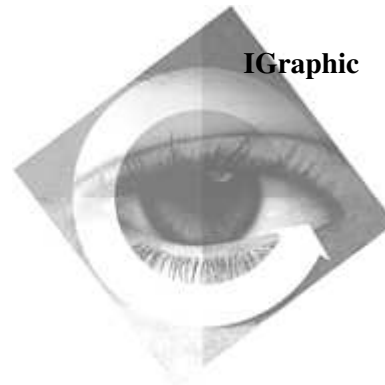
IBase		
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ELocation

ELocation {
 kInside, kOutside, kTopLeft, kTopMiddle, kTopRight,
 kMiddleLeft, kMiddleRight, kBottomLeft, kBottomMiddle, kBottomRight
};



IGraphic

Derivation

IBase
IVBase
IGraphic

Inherited By

IG3PointArc	IGPie
IGArc	IGPolyline
IGBitmap	IGRectangle
IGEllipse	IGRegion
IGLine	IGString
IGList	

Header File

igraphic.hpp

Members

Member	Page	Member	Page
Constructor	351	resetTransformMatrix	348
boundingRect	344	rotateBy	349
contains	346	scaleBy	349
drawOn	344	setGraphicBundle	345
graphicBundle	344	setHitSelectable	346
hasGraphicBundle	345	setHitSelected	346
hasTransformMatrix	348	setId	347
id	347	setTransformMatrix	349
isHitSelectable	346	setTransformMethod	349
isHitSelected	346	transformMatrix	350
operator !=	351	transformMethod	350
operator =	352	translateBy	350
operator ==	351	~IGraphic	347
removeGraphicBundle	345		

The IGraphic class is an abstract base class that provides common behavior for all two-dimensional graphic classes. The common behavior includes bounding rectangle information, accessing bundle attributes, transforming a graphic object, detecting if a graphic object has been selected (hit testing), and identifying a graphic object.

An IGraphic object can optionally contain an IGraphicBundle and an ITransformMatrix. You can set a graphic bundle on a graphic object so that the attributes set in the bundle are used when you draw a graphic object. If you use any of the world transform functions or set a transform matrix on a graphic object, then the transform matrix contained in the graphic object is used when you draw a graphic object.

IGraphic

The drawOn function is a pure virtual function that is overridden by all classes derived from IGraphic. This function provides the drawing operations necessary for rendering a graphic object on a device.

Public Functions

Bounding Rectangle

Use these members to determine the smallest rectangle that contains the graphic object. The graphic object's current world space transform and attribute bundle are used to determine the bounding rectangle.

boundingRect

Determines the smallest rectangle enclosing the graphic object. Graphic bundle attributes and the graphic object's current world transform matrix are used to determine the bounding rectangle.

virtual IRectangle boundingRect(IGraphicContext& graphicContext);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Drawing

Use these members to render a graphic object on a device. Derived classes must override drawOn.

drawOn

Provides the necessary drawing instructions for rendering the graphic object. This is a pure virtual function that must be overridden by all classes derived from IGraphic.

virtual IGraphic& drawOn(IGraphicContext& graphicContext) = 0;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Graphic Bundles

Use these members to query, set, and remove the drawing attributes that you use when drawing the graphic object.

graphicBundle

Returns a copy of the graphic bundle currently set for this graphic object. If you do not set a graphic bundle for the graphic object, a graphic bundle with no attributes is returned.

IGraphic

```
virtual IGraphicBundle  
    graphicBundle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

hasGraphicBundle

Returns true if you set a graphic bundle on this graphic object.

```
virtual IBase::Boolean  
    hasGraphicBundle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

removeGraphicBundle

Removes the current graphic bundle. When you draw this graphic object, the attributes set in the graphic context are used instead of the attributes that were set in the graphic bundle.

```
virtual IGraphic&  
    removeGraphicBundle();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setGraphicBundle

Makes a copy of the argument graphic bundle. The attributes you set in the bundle are used when you draw the graphic object. If this graphic object already has a graphic bundle set, the previous graphic bundle is deleted.

```
virtual IGraphic&  
    setGraphicBundle( const IGraphicBundle& graphicBundle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Hit Testing

Hit testing is the process of determining which graphic objects, if any, intersect an area of interest. When you want to select an area of interest on the screen, you usually move the pointer to the area of interest and signal (by clicking the mouse, for example) that this is the graphic object you want. The *area of interest* is a small rectangle centered at the hit point whose size is determined by the hit aperture size.

Use the contains member to determine if a point lies on or within the graphic object.

Use the remaining members when you are performing hit testing using the hit-testing members provided by the IGList class. When an object is determined to be under a point, its setHitSelected member is called with a true argument. Typically, you do not need to set or check the hit selection state of a graphic object because the IGList class returns the graphic objects under the specified point. You can, however, control whether you want a graphic object contained in an IGList to be selectable by calling setSelectable with a false argument. In this case, this graphic object is not considered when hit testing is performed.

IGraphic

contains

Returns true if the point lies on or within the object. This function uses drawOn(IGraphicContext&) to determine if the object contains the point.

virtual IBase::Boolean contains(IGraphicContext& graphicContext, const IPoint& point);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------



WindowsNT and Windows95 do not support this virtual function on those derived classes that have undergone unsupported graphical transformations (scaling or rotation).

isHitSelectable

Returns true if you want the graphic object to be selectable when hit testing is performed on an IGList.

virtual IBase::Boolean isHitSelectable() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

isHitSelected Return trues if setHitSelected was set to true and if isHitSelectable is true.

virtual IBase::Boolean isHitSelected() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

setHitSelectable

Indicates whether this graphic object is selectable. This is useful when performing hit testing. Set setSelectable to false for all objects that you do not want or need to be selected. The default is true. If you set setSelected to false, the IGList class does not attempt to determine if a point lies within this graphic object.

virtual IGraphic& setHitSelectable(IBase::Boolean hitSelectable = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

setHitSelected

Indicates that this graphic object is selected. The default is false. This function is used by the IGList class to mark the graphic objects that have been selected. Before performing hit testing using the functions provided in IGList, each graphic object's hit selection state is reset to false.

virtual IGraphic& setHitSelected(IBase::Boolean hitSelected = false);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IGraphic

Object Identifier

Use these members to associate an identifier with a graphic object. This is particularly useful when this object is part of an IGList (or any other collection) and you want to retrieve it later. By default, all object identifiers are zero.

id Returns the graphic object's identifier.

```
virtual unsigned long  
id() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setId Sets the graphic object's identifier.

```
virtual IGraphic&  
setId( unsigned long identifier );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You cannot construct, destruct, or copy an object of this class directly.

~IGraphic

```
virtual  
~IGraphic();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

World Space Transformation

A *transformation* is an operation performed on a graphic object that changes the object in one of four ways: translation, rotation, scaling, and shearing. Transformations enable an application to control the location, orientation, size, and shape of graphic objects on an output device.

The transformation of graphic objects can be conceptually divided into a series of distinct transformations applied from one logical stopping point to another. Coordinate spaces are used as a method of conceptualizing these logical stopping points. The coordinate spaces are concepts used to explain and manipulate the transformation process.

The *world coordinate space* is where most drawing coordinates are specified. If you picture the presentation space as a blank canvas on which to draw, the world coordinate space is a Cartesian grid that provides a reference scale for what is being drawn.

The components of a picture defined in world coordinate space are often defined to a scale convenient to only that component. Applications also can define each component, or *subpicture*, starting at the origin (0,0). This enables applications to define the scale of a subpicture and the location of the subpicture separately.

IGraphic

After subpictures are defined in world coordinate space, they undergo a process called transformation and then appear on an output device. If an application has not specifically applied a transformation on a subpicture, by default the identity transform is applied, which, in effect, makes no change to the subpicture.

Both the OS/2 operating system and Windows provide multiple coordinate spaces and use transformations to move subpictures between the coordinate spaces. The transformations you apply to a graphic object transform an object's data points from one coordinate space to another. In the OS/2 operating system, the graphic object's transform matrix is used to map from world coordinate space to model coordinate space. In Windows, the graphic object's transform matrix is used to map from world coordinate space to page coordinate space.

The transformation matrix does not affect an object's data points. The matrix is used to map the object's data points from world coordinate space when you draw the graphic object or perform hit testing on the graphic object.

When you draw a graphic object, the object's transform matrix is applied to the graphic context's current world transform matrix using the transform method.

By default, an object's transform matrix replaces the graphic context's current transform matrix before drawing and restores the previous transform matrix after drawing. You can provide an accumulative effect on transform matrixes by using the leftMultiply or rightMultiply transform method.

The leftMultiply transform method premultiplies the graphic object's transform matrix with the graphic context's world transform matrix before drawing the graphic object.

The rightMultiply transform method postmultiplies the graphic object's transform matrix with the graphic context's world transform matrix before drawing the graphic object.

hasTransformMatrix

Returns true if the graphic object's transform matrix is not the identity transform matrix.

<code>virtual IBase::Boolean</code>	Win	PM	Motif
<code>hasTransformMatrix() const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

resetTransformMatrix

Sets the graphic object's transform matrix to the identity transform matrix.

<code>virtual IGraphic&</code>	Win	PM	Motif
<code>resetTransformMatrix();</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphic

rotateBy Rotates the object by the angle, specified in degrees, around the point specified.

```
virtual IGraphic&
    rotateBy( double angle,
              const IPoint& point = IPoint ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Win Windows95 does not support this virtual function when the derived classes from IGraphic are IGellipse, IGarc, IG3PointArc, IGPie, IGChord, IGString, and IGBitmap. Although a rounded-corner IGRectangle can be rotated in Windows95, the rounding effect will be lost as a result. WindowsNT does not support this virtual function when the derived class from IGraphic is IGString using a bitmap font.

Exceptions	
IAccessError	The system could not perform the rotate operation. Make sure the object is not being accessed concurrently on another thread.

scaleBy Scales the object using the x and y factors around the point specified.

```
virtual IGraphic&
    scaleBy( double xScale,
             double yScale,
             const IPoint& point = IPoint ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Win WindowsNT and Windows95 do not support this virtual function when the derived class from IGraphic is IGString using a bitmap font.

Exceptions	
IAccessError	The system could not perform the scale operation. Make sure the object is not being accessed concurrently on another thread.

setTransformMatrix

Replaces the graphic object's transform matrix with the argument transform matrix. A copy of the argument transform matrix is created and the previous transform matrix is deleted.

```
virtual IGraphic&
    setTransformMatrix(
        const ITransformMatrix& transformMatrix);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setTransformMethod

Sets the method used to combine the graphic object's transform matrix with the existing graphic context's world transform matrix.

IGraphic

```
virtual IGraphic&
    setTransformMethod(
        TransformMethod transformMethod = rightMultiply);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

transformMatrix

This function returns the graphic object's transform matrix. If no transformations operations have been called, then the identity matrix is returned.

```
const ITransformMatrix
    transformMatrix() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

transformMethod

Returns the method used to combine world transform matrixes.

```
IGraphic::TransformMethod
    transformMethod() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

translateBy Moves the object by the amount specified in the IPoint.

```
virtual IGraphic&
    translateBy( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
IAccessError	The system could not perform the translate operation. Make sure the object is not being accessed concurrently on another thread.	

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

IGraphic

Comparisons

Use these members to compare two IGraphic objects.

operator != Returns true if the graphics are not identical (includes the graphic bundle and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGraphic& graphic) const;	Y	Y	N

operator == Returns true if the graphics are identical (includes the graphic bundle and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGraphic& graphic) const;	Y	Y	N

Constructors

You cannot construct, destruct, or copy an object of this class directly.

IGraphic

1	IGraphic();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use the default constructor when you want to create an IGraphic object that does not contain an attribute bundle and, with its transformation matrix, is initialized to the identity transform.

2	IGraphic(const IGraphicBundle& graphicBundle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use this function when you want to create an IGraphic object from an IGraphicBundle (p. 353) object. The IGraphic object makes a copy of the argument IGraphicBundle object and initializes its transformation matrix to the identity transform.

graphicBundle

An IGraphicBundle object. The IGraphic object uses the attributes set in the graphic bundle when drawing.

3	IGraphic(const IGraphic& graphic);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

IGraphic

Use this function when you want to create an IGraphic object from another IGraphic object.

graphic Reference to another class derived from IGraphic.

operator = Use this function when you want to assign an IGraphic object to another IGraphic object.

```
IGraphic&
operator =( const IGraphic& graphic );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	

TransformMethod

```
TransformMethod {
    replace,
    rightMultiply,
    leftMultiply
};
```

The transform method allows you to control the method the graphic object's transformation matrix is combined with the graphic context's current world transform matrix.

IGraphicBundle



IGraphicBundle

Derivation IBase
IGraphicBundle

Inherited By None.

Header File igbundle.hpp

Members	Member	Page	Member	Page
	Constructor	358	operator =	359
	backgroundColor	356	operator ==	358
	backgroundMixMode	355	patternOrigin	362
	backwardDiag1	370	penColor	356
	backwardDiag2	370	penEndingStyle	367
	dense1	371	penJoiningStyle	369
	dense2	371	penPattern	364
	dense3	371	penType	365
	dense4	371	penWidth	366
	dense5	372	resetBackgroundColor	357
	dense6	372	resetBackgroundMixMode	355
	dense7	372	resetDrawOperation	359
	dense8	372	resetFillColor	357
	drawOperation	359	resetFillPattern	364
	fillColor	356	resetMixMode	361
	filled	373	resetPatternOrigin	362
	fillPattern	363	resetPenColor	357
	forwardDiag1	373	resetPenEndingStyle	368
	forwardDiag2	373	resetPenJoiningStyle	369
	halfTone	373	resetPenPattern	364
	hasBackgroundColor	356	resetPenType	366
	hasBackgroundMixMode	355	resetPenWidth	366
	hasDrawOperation	359	setBackgroundColor	357
	hasFillColor	356	setBackgroundMixMode	355
	hasFillPattern	363	setDrawOperation	360
	hasMixMode	361	setFillColor	357
	hasPatternOrigin	361	setFillPattern	364
	hasPenColor	356	setMixMode	361
	hasPenEndingStyle	367	setPatternOrigin	362
	hasPenJoiningStyle	369	setPenColor	357
	hasPenPattern	363	setPenEndingStyle	368
	hasPenType	365	setPenJoiningStyle	369
	hasPenWidth	366	setPenPattern	364
	hatched	374	setPenType	366
	hatchedDiag	374	setPenWidth	367
	hollow	374	vertical	374
	horizontal	374	~IGraphicBundle	359
	mixMode	361		
	operator !=	358		

IGraphicBundle

The IGraphicBundle class allows you to set drawing attributes for graphic objects (such as IGLine, IGArc, and IGPolyline). There are cases where this class allows you to set an attribute for a graphic object that does not support the attribute. For example, you can set a fill color for a graphic bundle object. You then set the graphic bundle object on an IGLine object. The fill color has no effect when you draw the line because lines do not have fill colors.

The graphic bundle class allows you to selectively change drawing attributes from the drawing attributes currently set for a graphic context. If a graphic object does not contain a graphic bundle, the current graphic context drawing attributes are used when you draw a graphic object. If you call a set attribute function of a graphic bundle object and set the graphic bundle on a graphic object, this graphic bundle's attributes override the drawing attributes for the graphic context when you draw the graphic object. For drawing attributes that you have not changed using a set attribute function, the current drawing attributes set in the graphic context are used when you draw a graphic object.

Each drawing attribute has four functions associated with it: set, query, has, and reset. You use a set function to set a drawing attribute on the graphic bundle. You use a query function to obtain a drawing attribute set on a graphic bundle. You use a has function to determine if a drawing attribute is set on a graphic object. And, there is a reset function to use the current graphic context drawing attribute. Thus, the drawing attribute set in the graphic bundle is no longer used when drawing a graphic object.

Public Functions

Background Mix Mode

Use these members to govern how the background color of a graphic object is combined with any existing drawing. The *background* of an object is the area between fill lines when you draw a closed figure using a nonsolid fill pattern, the area of a character box when you draw a graphic string, or the 0 bits when you draw a 1-bit-per-pel bitmap. The default background mix mode is backLeaveAlone. This background mix mode has no effect on the background color when you draw a graphic object. The following is a list of the supported background mix modes:

- | | |
|----------------|--|
| backLeaveAlone | The resulting color is determined by the display surface. |
| backOverPaint | The resulting color is determined by the background color. |
| backOr | The resulting color is determined by the bitwise OR of the background color and the color of the drawing surface. |
| backXor | The resulting color is determined by the bitwise XOR of the background color and the color of the drawing surface. |

IGraphicBundle

backgroundMixMode

Returns the background mix mode. If you have not set a background mix mode on the graphic bundle, the default background mix mode of the graphic context is returned.

BackgroundMixMode	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backgroundMixMode() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

hasBackgroundMixMode

Returns true if you have set the background mix mode on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasBackgroundMixMode() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetBackgroundMixMode

Does not use the background mix mode set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context background mix mode is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetBackgroundMixMode();	<i>Y</i>	<i>Y</i>	<i>N</i>

setBackgroundMixMode

Sets the background mix mode.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setBackgroundMixMode(BackgroundMixMode backgroundMixMode = backLeaveAlone);	<i>Y</i>	<i>Y</i>	<i>N</i>

Color Attributes

Use these members to control the use of pen, fill, and background colors for a graphic bundle. The pen color specifies the color used when you draw lines or when you frame a closed figure. You use the fill color to specify the color of the interior of a closed figure, such as an ellipse.

The background color is used when you draw a closed figure, a graphic string, or a 1-bit-per-pel bitmap. If you set the fill pattern to IGraphicBundle::filled, the background color does not have an effect. If, however, you specify a fill pattern of IGraphicBundle::vertical, for example, the color between the vertical lines is the current background color. The default background mix mode is IGraphicBundle::backLeaveAlone; it specifies that the background color is not drawn. This means that for a nonsolid fill pattern, the drawing surface color shows through. The

IGraphicBundle

background color for nonsolid fill patterns appears only if the background mix mode is changed to IGraphicBundle::backOverPaint.

When you draw a graphic string, you can use the background color to specify the color of the character box that each character is drawn in.

When you draw a 1-bit-per-pel bitmap, the background color is used when drawing the bits that have a 0 value, whereas the pen color is used to draw the bits that have a 1 value.

backgroundColor

Returns the background color. If you have not set a background color, the default background color of the graphic context is returned. Unless you change it, the default is black.

IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backgroundColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

fillColor

Returns the fill color. If you have not set a fill color, the default fill color of the graphic context is returned. Unless you change it, the default is black.

IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
fillColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

hasBackgroundColor

Returns true if you have set the background color on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasBackgroundColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

hasFillColor

Returns true if you have set the fill color on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasFillColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

hasPenColor

Returns true if you have set the pen color on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasPenColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

penColor

Returns the pen color. If you have not set pen color, the default pen color of the graphic context is returned. Unless you change it, the default is black.

IGraphicBundle

IColor
penColor() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

resetBackgroundColor

Does not use the background color set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context background color is used.

IGraphicBundle&
resetBackgroundColor();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

resetFillColor

Does not use the fill color set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context fill color is used.

IGraphicBundle&
resetFillColor();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

resetPenColor

Does not use the pen color set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen color is used.

IGraphicBundle&
resetPenColor();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setBackgroundColor

Sets the current background color.

IGraphicBundle&
setBackgroundColor(const IColor& backColor);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setFillColor

Sets the current fill color.

IGraphicBundle&
setFillColor(const IColor& fillColor);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setPenColor

Sets the current pen color.

IGraphicBundle&
setPenColor(const IColor& penColor);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicBundle

Comparisons

Use these members to compare two IGraphicBundle objects.

operator != Returns true if the drawing attributes are not identical.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGraphicBundle& graphicBundle);	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if all drawing attributes are identical

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGraphicBundle& graphicBundle);	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGraphicBundle

1	IGraphicBundle(const IGraphicBundle& graphicBundle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGraphicBundle object from another IGraphicBundle object. The drawing attributes set in the graphic bundle object are set in the new graphic bundle object.

graphicBundle An IGraphicBundle object.

2	IGraphicBundle();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use the default construct to create an IGraphicBundle object. Initially, no drawing attributes are set.

3	IGraphicBundle(const IGraphicContext& graphicContext);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGraphicBundle object from a IGraphicContext object. All drawing attributes in the graphic bundle object are set to the current graphic context drawing attributes.

graphicContext An IGraphicContext object.

IGraphicBundle

operator = Use this function to assign an IGraphicBundle object to another IGraphicBundle object.

IGraphicBundle& operator =(const IGraphicBundle& graphicBundle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

~IGraphicBundle

~IGraphicBundle();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--------------------	-----------------	----------------	-------------------

Draw Operations

Use these members to control the method used to draw a closed graphic object (such as IGPie, IGChord, IGPolygon, IGRectangle, or IGEllipse). The draw operation's values are as follows:

fill	Only the interior of the graphic object is drawn.
frame	Only the frame of the graphic object is drawn.
fillAndFrame	The interior and the frame of the graphic object are drawn.

drawOperation

Returns the draw operation. If you have not set a draw operation, the default draw operation of the graphic context is returned.

DrawOperation drawOperation() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

hasDrawOperation

Returns true if you have set the draw operation on the graphic bundle.

IBase::Boolean hasDrawOperation() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

resetDrawOperation

Does not use the draw operation set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context draw operation is used.

IGraphicBundle& resetDrawOperation();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IGraphicBundle

setDrawOperation

Sets the draw operation. This attribute only affects closed graphic objects.

IGraphicBundle& setDrawOperation(DrawOperation drawOperation = fillAndFrame);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Mix Mode

Use these member to govern how the pen and fill colors of a graphic object are combined with any existing drawing. The default mix mode is overPaint. When you draw a graphic object with this mix mode, the pen and fill colors used to draw the graphic object paint over any existing drawing in the same area on the drawing surface. The following is a list of the supported mix modes.

Note: The description of each mix mode refers to the pen color, but the same description applies to the fill color as well.

bitOr	Resulting color is determined by a bitwise OR of the pen color and the drawing surface.
overPaint	Resulting color is determined by pen color. This is the default for the pen mix attribute.
bitExclusiveOr	Resulting color is determined by a bitwise XOR of the pen color and the drawing surface. This mode enables graphic objects to be drawn so that they can be removed easily by simply drawing them a second time using the xor mix mode. This is useful for graphic animation when an application must move a graphic object and completely restore the graphic objects that it originally overlapped.
leaveAlone	Resulting color is determined by the drawing surface.
bitAnd	Resulting color is determined by a bitwise AND of the pen color and the drawing surface.
subtract	Resulting color is determined by inverting the pen color and performing a bitwise AND with the drawing surface.
maskSourceNot	Resulting color is determined by inverting the surface color and performing a bitwise AND with the pen color.
black	Resulting color is black.
notMergeSource	Resulting color is the inverse of the or mix mode.
notXorSource	Resulting color is the inverse of the xor mix mode.
invert	Resulting color is the inverse of the drawing surface.

IGraphicBundle

mergeSourceNot	Resulting color is determined by a bitwise OR of the pen color and the inverse color of the drawing surface.
notCopySource	Resulting color is the inverse of the pen color.
mergeNotSource	Resulting color is determined by the inverse of the pen color and by performing a bitwise AND of the drawing surface.
notMaskSource	Resulting color is the inverse of the "and" pen mix.
white	Resulting color is white.

hasMixMode Returns true if you have set the mix mode on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasMixMode() const;	Y	Y	N

mixMode Returns the mix mode. If you have not set a mix mode on the graphic bundle, the default mix mode of the graphic context is returned.

MixMode	<u>Win</u>	<u>PM</u>	<u>Motif</u>
mixMode() const;	Y	Y	N

resetMixMode

Does not use the mix mode set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context mix mode is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetMixMode();	Y	Y	N

setMixMode Sets the mix mode.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setMixMode(MixMode mixMode = overPaint);	Y	Y	N

Pattern Origin

Use these members to define a point from which the pen and fill patterns spread horizontally and vertically. The lower-left corner is aligned on this point and is expressed in world coordinate space.

hasPatternOrigin

Returns true if you have set the pattern origin point on the graphic bundle.

IGraphicBundle

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasPatternOrigin() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

patternOrigin Returns the origin point of the pen and fill patterns. If the pattern origin has not been set on the graphic bundle, the default pattern origin point of the graphic context is returned.

IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
patternOrigin() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetPatternOrigin

Does not use the pattern origin point set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pattern origin point is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetPatternOrigin();	<i>Y</i>	<i>Y</i>	<i>N</i>

setPatternOrigin

Sets the origin point to which the pen and fill patterns map.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPatternOrigin(const IPoint& point = IPoint (0 , 0));	<i>Y</i>	<i>Y</i>	<i>N</i>

Pen and Fill Patterns

Use these members to control the pattern used when you draw lines that have a pen width greater than 1 or when you fill a closed figure. See the following figure for an example of pen and fill pattern.

IGraphicBundle

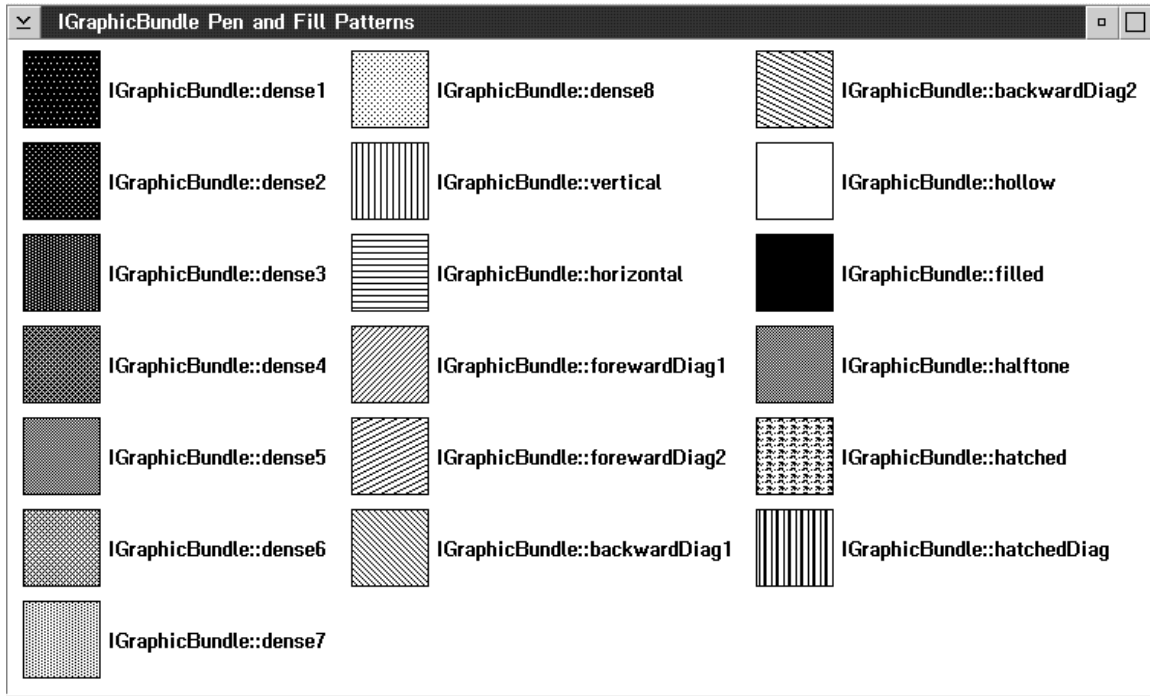


Figure 1. Pen and Fill Patterns

fillPattern Returns the fill pattern. If you have not set a fill pattern on the graphic bundle, the default fill pattern of the graphic context is returned.

```
unsigned long  
fillPattern() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

hasFillPattern

Returns true if you have set the fill pattern on the graphic bundle.

```
IBase::Boolean  
hasFillPattern() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

hasPenPattern

Returns true if you have set the pen pattern on the graphic bundle.

```
IBase::Boolean  
hasPenPattern() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicBundle

penPattern Returns the pen pattern. If you have not set a pen pattern on the graphic bundle, the default pen pattern of the graphic context is returned.

unsigned long penPattern() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--------------------------------------	-----------------	----------------	-------------------

resetFillPattern

Does not use the fill pattern set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context fill pattern is used.

IGraphicBundle& resetFillPattern();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

resetPenPattern

Does not use the pen pattern set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen pattern is used.

IGraphicBundle& resetPenPattern();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

setFillPattern Sets the pattern used to fill a closed figure. You can also specify a bitmap handle that is used to fill a closed figure. The bitmap must be 8-by-8 and 1-bit-per-pel.

IGraphicBundle& setFillPattern(unsigned long fillPattern = filled);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

setPenPattern

Sets the pattern used for the pen. The pen width must be greater than 1 for the pen pattern to have an effect.

IGraphicBundle& setPenPattern(unsigned long penPattern = filled);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Pen Type

Use these members to define the way lines, arcs, polylines, and the frame on closed figures are drawn. The pen type can be solid, a series of dashes, a series of dots, or a combination of dashes and dots. If the pen width is greater than 1, graphic objects are drawn using a pen type of solid, regardless of the pen type set. The following is a list of the supported pen types:

solid An uninterrupted line is drawn. This is the default pen type.

IGraphicBundle

dot	A series of dots are drawn.
shortDash	A series of short dashes are drawn.
dashDot	A series of dashes followed by a dot are drawn.
longDash	A series of long dashes are drawn.
dashDoubleDot	A series of a dashes followed by two dots are drawn.
alternate	Alternate pels are drawn.
invisible	The pen is not visible.

See the following figure for an example of pen types.

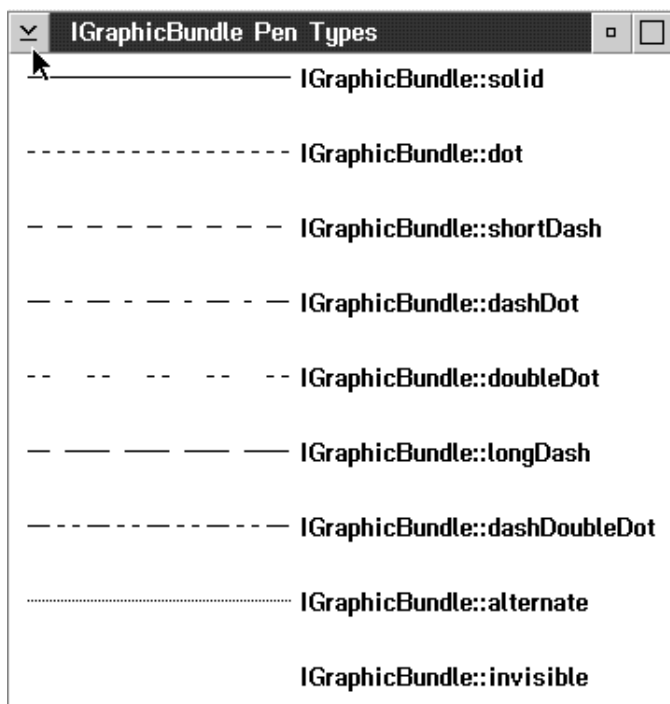


Figure 2. Pen Types

hasPenType Returns true if you have set the pen type on the graphic bundle.

```
IBase::Boolean
hasPenType() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

penType Returns the pen type. If you have not set the pen on the graphic bundle, the default pen type of the graphic context is returned.

IGraphicBundle

PenType	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penType() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetPenType

Does not use the pen type set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen type is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetPenType();	<i>Y</i>	<i>Y</i>	<i>N</i>

setPenType

Sets the current pen type. The pen width must be 1 for a pen type other than solid to be used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPenType(PenType penType = solid);	<i>Y</i>	<i>Y</i>	<i>N</i>

Pen Width

Use these members to govern the width of the pen when you draw lines or the width of the frame when you draw a closed figure. The width specified is in world coordinate space and is subject to world space transforms. If you specify a pen width greater than 1, the pen type is always solid.

hasPenWidth Returns true if you set the pen width on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasPenWidth() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

penWidth

Returns the pen width. If you have not set the pen width on the graphic bundle, the default pen width of the graphic context is returned.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penWidth() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetPenWidth

Does not use the pen width set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen width is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetPenWidth();	<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicBundle

setPenWidth Sets the width of the pen. If the value is greater than 1, the pen type is always solid.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPenWidth(unsigned long lineWidth = 1);	Y	Y	N

Pen-Ending Style

Use these members to control the shape of the unattached end of a line. The pen width must be greater than 1 for the pen-ending style to have an effect. The following is a list of the supported pen-ending styles:

flat	Lines drawn with this style are flat, finishing flush with the end points of the line.
square	Lines drawn with this style are flat, finishing past the end of the line. The line is extended by a distance that is equal to half the pen width.
round	Lines drawn with this style are rounded, finishing past the end of the line. The rounded end is constructed by drawing a circle whose radius is half the pen width.

See the following figure for an example of pen-ending styles.

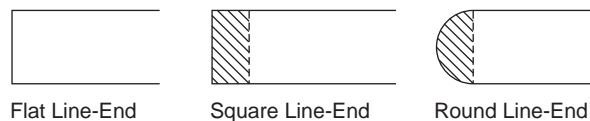


Figure 3. Pen-Ending Styles

hasPenEndingStyle

Returns true if you have set the pen-ending style on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasPenEndingStyle() const;	Y	Y	N

penEndingStyle

Returns the pen-ending style. If you have not set the pen-ending style on the graphic bundle, the default pen-ending style of the graphic context is returned.

PenEndingStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penEndingStyle() const;	Y	Y	N

IGraphicBundle

resetPenEndingStyle

Does not use the pen-ending style set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen-ending style is used.

```
IGraphicBundle&  
resetPenEndingStyle();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setPenEndingStyle

Sets the pen-ending style. The pen width must be greater than 1 for the pen-ending style to have an effect.

```
IGraphicBundle&  
setPenEndingStyle( PenEndingStyle penEndingStyle = flat );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Pen-Joining Style

Use these members to control the shapes formed by two intersecting lines. The pen width must be greater than 1 for the pen-joining style to have an effect. The following is a list of the supported pen-joining styles:

bevel	The joint has a diagonal corner.
round	The joint has a rounded corner.
miter	The joint has a 90-degree-angled corner.

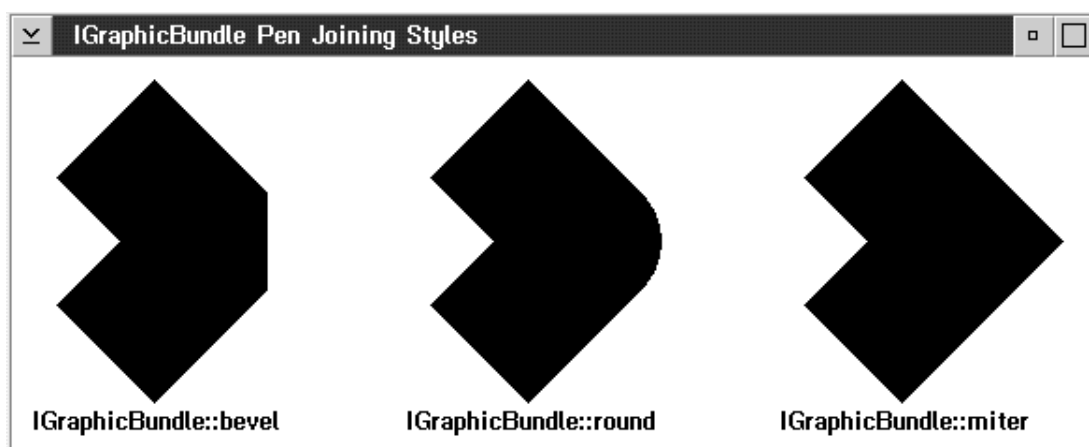


Figure 4. Pen-Joining Styles

IGraphicBundle

hasPenJoiningStyle

Returns true if you have set the pen-joining style on the graphic bundle.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasPenJoiningStyle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

penJoiningStyle

Returns the pen-joining style. If you have not set the pen-joining style on the graphic bundle, the default pen-joining style of the graphic context is returned.

PenJoiningStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penJoiningStyle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

resetPenJoiningStyle

Does not use the pen-joining style set in the graphic bundle when you draw graphic objects with this graphic bundle. The current graphic context pen-joining style is used.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetPenJoiningStyle();	<i>Y</i>	<i>Y</i>	<i>N</i>

setPenJoiningStyle

Sets the pen-joining style. The pen width must be greater than 1 for the pen-joining style to have an effect.

IGraphicBundle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPenJoiningStyle(PenJoiningStyle penJoiningStyle = bevel);	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Public Data

IGraphicBundle

Pen and Fill Patterns

Use these members to control the pattern used when you draw lines that have a pen width greater than 1 or when you fill a closed figure. See the following figure for an example of pen and fill pattern.

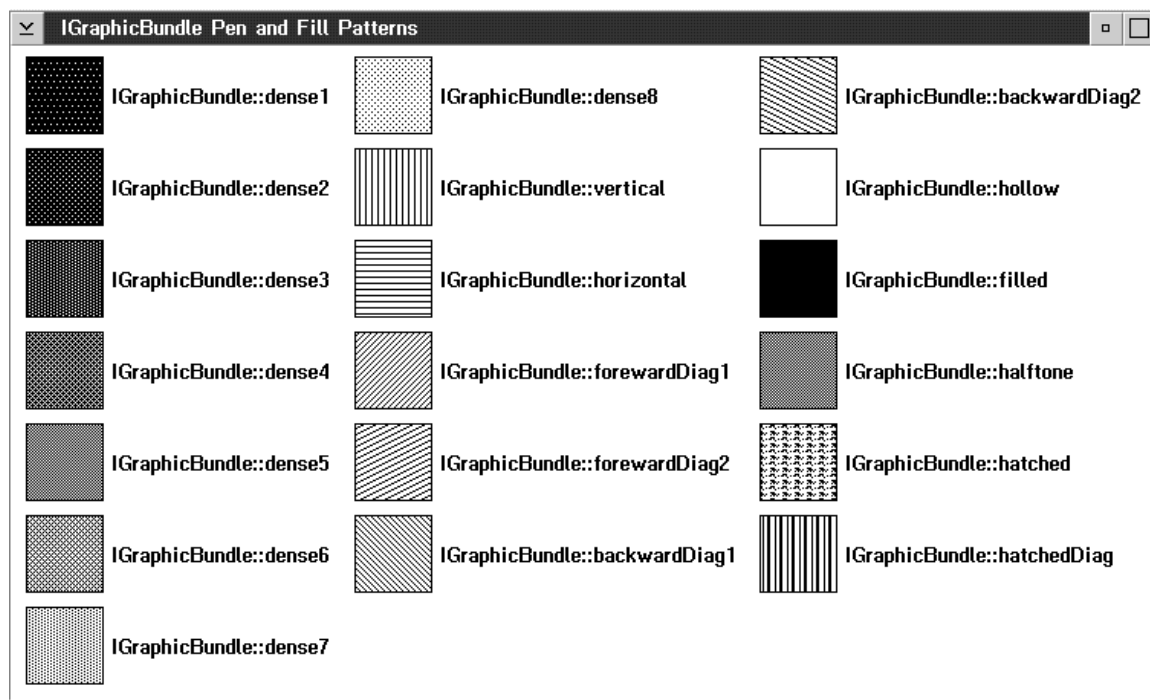


Figure 5. Pen and Fill Patterns

backwardDiag1

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
backwardDiag1;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N



This pattern is not currently supported for pens on Windows 95.

backwardDiag2

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

IGraphicBundle

```
static const unsigned long  
backwardDiag2;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense1

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
dense1;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense2

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
dense2;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense3

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
dense3;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense4

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

IGraphicBundle

static const unsigned long dense4;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense5

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long dense5;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense6

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long dense6;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense7

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long dense7;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

dense8

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

IGraphicBundle

```
static const unsigned long  
dense8;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

filled

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
filled;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

forewardDiag1

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
forewardDiag1;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is not currently supported for pens on Windows 95.

forewardDiag2

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
forewardDiag2;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

This pattern is not currently supported for pens on Windows 95.

halftone

Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

```
static const unsigned long  
halftone;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This pattern is provided by creating a pattern brush. As such, the background mix mode IGraphicBundle::backLeaveAlone is not supported when using this pattern.

IGraphicBundle

This pattern is not currently supported for pens on Windows 95.

hatched Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long hatched;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------



This pattern is not currently supported for pens on Windows 95.

hatchedDiag Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long hatchedDiag;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------



This pattern is not currently supported for pens on Windows 95.

hollow Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long hollow;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

horizontal Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long horizontal;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------



This pattern is not currently supported for pens on Windows 95.

vertical Pattern used when you draw a line that has a width greater than 1 or when you fill a figure.

static const unsigned long vertical;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------



This pattern is not currently supported for pens on Windows 95.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

BackgroundMixMode

```
BackgroundMixMode {
    backLeaveAlone,
    backOverPaint,
    backOr,
    backXor
};
```

These functions govern how the background color of a graphic object is combined with any existing drawing. The *background* of an object is the area between fill lines when you draw a closed figure using a nonsolid fill pattern, the area of a character box when you draw a graphic string, or the 0 bits when you draw a 1-bit-per-pel bitmap. The default background mix mode is backLeaveAlone. This background mix mode has no effect on the background color when you draw a graphic object. The following is a list of the supported background mix modes.

backLeaveAlone The resulting color is determined by the display surface.
backOverPaint The resulting color is determined by the background color.
backOr The resulting color is determined by the bitwise OR of the background color and the color of the drawing surface.
backXor The resulting color is determined by the bitwise XOR of the background color and the color of the drawing surface.

DrawOperation

```
DrawOperation {
    fill,
    frame,
    fillAndFrame
};
```

These functions control the method used to draw a closed graphic object (such as IGPie, IGChord, IGPolygon, IGRectangle, or IGEllipse). The draw operation's values are:

fill Only the interior of the graphic object is drawn.
frame Only the frame of the graphic object is drawn.
fillAndFrame The interior and the frame of the graphic object are drawn.

MixMode

```
MixMode {
    bitOr,                overPaint,                bitExclusiveOr,
    leaveAlone,           bitAnd,                subtract,
    maskSourceNot,        black,                notMergeSource,
    notXorSource,          invert,                mergeSourceNot,
```

IGraphicBundle

```
notCopySource,      mergeNotSource,      notMaskSource,  
white,              or = bitOr,          xor = bitExclusiveOr,  
and = bitAnd  
};
```

The functions govern how the pen and fill colors of a graphic object are combined with any existing drawing. The default mix mode is overPaint. When you draw a graphic object with this mix mode, the pen and fill colors used to draw the graphic object paints over any existing drawing in the same area on the drawing surface. The following is a list of the supported mix modes.

Note: The description of each mix mode refers to the pen color, but the same description applies to the fill color as well.

bitOr	Resulting color is determined by a bitwise OR of the pen color and the drawing surface.
overPaint	Resulting color is determined by pen color. This is the default for the pen mix attribute.
bitExclusiveOr	Resulting color is determined by a bitwise XOR of the pen color and the drawing surface. This mode enables graphic objects to be drawn so that they can be removed easily by simply drawing them a second time using the bitExclusiveOr mix mode. This is useful for graphic animation when an application must move a graphic object and completely restore the graphic objects that it originally overlapped.
leaveAlone	Resulting color is determined by the drawing surface.
bitAnd	Resulting color is determined by a bitwise AND of the pen color and the drawing surface.
subtract	Resulting color is determined by inverting the pen color and performing a bitwise AND with the drawing surface.
maskSourceNot	Resulting color is determined by inverting the surface color and performing a bitwise AND with the pen color.
black	Resulting color is black.
notMergeSource	Resulting color is the inverse of the or mix mode.
notXorSource	Resulting color is the inverse of the xor mix mode.
invert	Resulting color is the inverse of the drawing surface.
mergeSourceNot	Resulting color is determined by a bitwise OR of the pen color and the inverse color of the drawing surface.
notCopySource	Resulting color is the inverse of the pen color.

IGraphicBundle

mergeNotSource	Resulting color is the determined by the inverse of the pen color and performing a bitwise AND of the drawing surface.
notMaskSource	Resulting color is the inverse of the and pen mix.
white	Resulting color is white.

PenEndingStyle

```
PenEndingStyle {  
    flat,  
    square,  
    rounded  
};
```

These functions control the shape of the unattached end of a line. The pen width must be greater than 1 for the pen-ending style to have an effect. The following is a list of the supported pen-ending styles.

flat	Lines drawn with this style are flat, finishing flush with the end points of the line.
square	Lines drawn with this style are flat, finishing past the end of the line. The line is extended by a distance that is equal to half the pen width.
round	Lines drawn with this style are rounded, finishing past the end of the line. The rounded end is constructed by drawing a circle whose radius is half the pen width.



The pen-ending style is not currently supported on Windows 95.

PenJoiningStyle

```
PenJoiningStyle {  
    bevel,  
    round,  
    miter  
};
```

These functions control the shaped formed by two intersecting lines. The pen width must be greater than 1 for the pen-joining style to have an affect. The following is a list of the supported pen-joining styles.

bevel	The joint has a diagonal corner.
round	The joint has a rounded corner.
miter	The joint has a 90 degree angled corner.



The pen-joining style is not currently supported on Windows 95.

IGraphicBundle

PenType

```
PenType {  
    solid,          dot,          shortDash,    dashDot,        doubleDot,  
    longDash,       dashDoubleDot, alternate,    invisible  
};
```

These functions define the way lines, arcs, polylines, and the frame on closed figures are drawn. The pen type can be solid, a series of dashes, a series of dots, or a combination of dashes and dots. If the pen width is greater than 1, graphic objects are drawn using a pen type of solid regardless of the pen type set. The following is a list of the supported pen types.

<code>solid</code>	An uninterrupted line is drawn. This is the default pen type.
<code>dot</code>	A series of dots are drawn.
<code>shortDash</code>	A series of short dashes are drawn.
<code>dashDot</code>	A series of dash followed by a dot is drawn.
<code>longDash</code>	A series of long dashes are drawn.
<code>dashDoubleDot</code>	A series of a dash followed by two dots are drawn.
<code>alternate</code>	Alternate pels are drawn.
<code>invisible</code>	The pen is not visible.



Only the solid and invisible pen types are currently supported on Windows 95.

IGraphicContext

Derivation IBase
 IVBase
 IGraphicContext

Inherited By None.

Header File igrfctx.hpp

Members	Member	Page	Member	Page
	Constructor	382	penJoiningStyle	394
	addToBoundingRect	380	penPattern	394
	backgroundColor	392	penType	394
	backgroundMixMode	392	penWidth	394
	clearClipRegion	403	recoordinationHeight	402
	clipRegion	403	resetFont	398
	currentDrawingPosition	383	setBackgroundColor	394
	currentFont	398	setBackgroundMixMode	395
	defaultBackgroundColor	383	setClipRegion	403
	defaultBackgroundMixMode	384	setCurrentDrawingPosition	383
	defaultDrawOperation	384	setDefaultBackgroundColor	386
	defaultFillColor	384	setDefaultBackgroundMixMode	386
	defaultFillPattern	384	setDefaultDrawOperation	386
	defaultMixMode	384	setDefaultFillColor	386
	defaultPatternOrigin	385	setDefaultFillPattern	386
	defaultPenColor	385	setDefaultMixMode	387
	defaultPenEndingStyle	385	setDefaultPatternOrigin	387
	defaultPenJoiningStyle	385	setDefaultPenColor	387
	defaultPenPattern	385	setDefaultPenEndingStyle	387
	defaultPenType	385	setDefaultPenJoiningStyle	387
	defaultPenWidth	386	setDefaultPenPattern	387
	draw	389	setDefaultPenType	388
	drawOperation	392	setDefaultPenWidth	388
	fillColor	392	setDrawOperation	395
	fillPattern	392	setFillColor	395
	graphicBundle	392	setFillPattern	395
	handle	383	setFont	399
	hasFont	398	setGraphicBundle	396
	hitApertureSize	399	setHitApertureSize	400
	hitPoint	399	setHitPoint	400
	isAccumulatingBoundingRect	381	setMappingMode	401
	isHitTesting	399	setMixMode	396
	mappingMode	401	setPageSize	388
	mixMode	393	setPatternOrigin	396
	pageSize	388	setPenColor	396
	patternOrigin	393	setPenEndingStyle	397
	penColor	393	setPenJoiningStyle	397
	penEndingStyle	393	setPenPattern	397

IGraphicContext

Member	Page	Member	Page
setPenType	397	startHitTesting	400
setPenWidth	398	stopBoundaryAccumulation	381
setRecoordinationHeight	402	stopHitTesting	401
setViewPortRect	389	viewPortRect	389
setWorldTransformMatrix	404	worldTransformMatrix	404
startBoundaryAccumulation	381	~IGraphicContext	382

The IGraphicContext class renders graphic objects on a device. IGraphicContext contains the graphic state information used when drawing graphics. The graphic state includes the current attribute bundle, the current world space transform matrix, and the current clip region.

If you are familiar with the use of presentation spaces in Presentation Manager or device contexts in Windows, you can view IGraphicContext as a wrapper class for these data types.

The IGraphicContext class also contains a static collection of default drawing attributes that initializes the attributes of the graphic context when you create it. You can change any of these default drawing attributes so that any graphic context created subsequently will be initialized with these new drawing attributes.

You can at any time use other native graphic programming interfaces whose function has not been included in either this class or any of the IGraphic derived classes. You can do this by accessing either the presentation space handle or device context by using the handle member function as the first parameter to the graphic programming interface.

Public Functions

Boundary Accumulation

Boundary accumulation is the process of determining the smallest enclosing rectangle that contains one or more graphic object. Use the members to start and stop boundary accumulation and query a bounding rectangle of one or more graphic objects. Normally, you do not need to use these functions. Instead, use IGraphic::boundingRect to query the bounding rectangle of a graphic object.

addToBoundingRect

If the graphic context is currently accumulating boundary data, the graphic context's boundary data forms a union with the rectangle specified.

IGraphicContext

```
virtual IGraphicContext&
    addToBoundingRect( const IRectangle& rectangle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isAccumulatingBoundingRect

Returns true if the graphic context is currently accumulating boundary data.

```
virtual IBase::Boolean
    isAccumulatingBoundingRect() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This function is useful when you are creating a class which derives from IGraphic. In your classes drawOn member function you should call this function to determine if the graphic context is accumulating boundary data. If this function returns true, you should add the graphic objects bounding rectangle to the graphic context's boundary data using IGraphicContext::addToBoundingRect. If this function returns false, you should draw your graphic object.

startBoundaryAccumulation

Starts the boundary accumulation process; subsequent drawing of graphic objects are added to the bounding rectangle. Drawing graphic objects are not visible when you are accumulating boundary data.

```
virtual IGraphicContext&
    startBoundaryAccumulation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to start boundary accumulation. Refer to the exception text for specific error information.

stopBoundaryAccumulation

Stops the boundary accumulation process and returns the bounding rectangle of all drawing operations that were made since calling the startBoundaryAccumulation function. Subsequent drawing of graphic objects are visible.

```
virtual IRectangle
    stopBoundaryAccumulation();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to stop boundary accumulation. Refer to the exception text for specific error information.

Constructors

You can construct and destruct objects of this class.

IGraphicContext

IGraphicContext

1 IGraphicContext(const ISize& contextSize); Win PM Motif
Y Y N

Use this function to create an IGraphicContext object from an ISize object. This creates a memory graphic context and sets a bitmap into the graphic context with the dimensions specified in the ISize object. This allows you to draw into a bitmap. When you are finished drawing, you can create an IGBitmap object from the graphic context using the IGBitmap constructor, which takes an IGraphicContext object and an IRectangle object as parameters.

2 IGraphicContext(); Win PM Motif
Y Y N

Use the default constructor to create a graphic context. This creates a memory graphic context. A memory graphic context is used when you want to draw graphic objects in memory and subsequently create a bitmap from the drawn graphic objects.

3 IGraphicContext(const IPresSpaceHandle& presSpaceHandle); Win PM Motif
Y Y N

Use this member to construct an IGraphicContext object from an IPresSpaceHandle object. You would typically use this member to construct a IGraphicContext object from the IPresSpaceHandle object provided during paint events.

Note: If you construct an IGraphicContext object with this constructor, the IPresSpaceHandle is not destroyed when the IGraphicContext is deleted.

presSpaceHandle Handle of presentation space.

4 IGraphicContext(const IWindowHandle& windowHandle); Win PM Motif
Y Y N

Use this function to create an IGraphicContext object from an IWindowHandle object. A graphic context is created that is compatible with the screen device that is associated with the window handle.

windowHandle Window handle used to get a presentation space in the OS/2 operating system or device context in Windows.

~IGraphicContext

IGraphicContext

```
virtual
    ~IGraphicContext();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these member to use the graphic context with native programming interfaces.

handle Returns the presentation space handle wrapped by the graphic context.

```
IPresSpaceHandle
    handle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Current Drawing Position

Use these members to set and query the graphic context's current drawing position. These functions are useful when you are creating your own graphic object derived from IGraphic.

currentDrawingPosition

Returns the graphic context's current drawing position.

```
virtual IPoint
    currentDrawingPosition() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setCurrentDrawingPosition

Sets the graphic context's current drawing position to the point specified.

```
virtual IGraphicContext&
    setCurrentDrawingPosition( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the current drawing position. Refer to the exception text for specific error information.

Default Drawing Attributes

Use these members to set and query the default drawing attributes that are used to initialize the drawing attributes of a graphic context when a graphic context is created.

defaultBackgroundColor

Returns the default background color. Unless you change it, the default background color is black.

IGraphicContext

<code>static IColor defaultBackgroundColor();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultBackgroundMixMode

Returns the default background mix mode. Unless you change it, the default background mix mode is IGraphicBundle::backLeaveAlone.

<code>static IGraphicBundle::BackgroundMixMode defaultBackgroundMixMode();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultDrawOperation

Returns the default drawing operation. Unless you change it, the default drawing operation is IGraphicBundle::fillAndFrame.

<code>static IGraphicBundle::DrawOperation defaultDrawOperation();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultFillColor

Returns the default fill color. Unless you change it, the default fill color is black.

<code>static IColor defaultFillColor();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultFillPattern

Returns the default fill pattern. Unless you change it, the default fill pattern is IGraphicBundle::solid.

<code>static unsigned long defaultFillPattern();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultMixMode

Returns the default mix mode. Unless you change it, the default mix mode is IGraphicBundle::overPaint.

<code>static IGraphicBundle::MixMode defaultMixMode();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicContext

defaultPatternOrigin

Returns the default pattern origin point. Unless you change it, the default pattern origin is 0,0.

static IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPatternOrigin();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenColor

Returns the default pen color. Unless you change it, the default pen color is black.

static IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPenColor();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenEndingStyle

Returns the default pen-ending style. Unless you change it, the default pen-ending style is IGraphicBundle::flat.

static IGraphicBundle::PenEndingStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPenEndingStyle();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenJoiningStyle

Returns the default pen-joining style. Unless you change it, the default pen-joining style is IGraphicBundle::bevel.

static IGraphicBundle::PenJoiningStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPenJoiningStyle();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenPattern

Returns the default pen pattern. Unless you change it, the default pen pattern is IGraphicBundle::solid.

static unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPenPattern();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenType

Returns the default pen type. Unless you change it, the default pen type is IGraphicBundle::solid.

IGraphicContext

<code>static IGraphicBundle::PenType defaultPenType();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPenWidth

Returns the default pen width. Unless you change it, the default pen width is 1.

<code>static unsigned long defaultPenWidth();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultBackgroundColor

Sets the default background color.

<code>static void setDefaultBackgroundColor(const IColor& backgroundColor);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultBackgroundMixMode

Sets the default background mix mode.

<code>static void setDefaultBackgroundMixMode(IGraphicBundle::BackgroundMixMode backMixMode);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultDrawOperation

Sets the default drawing operation.

<code>static void setDefaultDrawOperation(IGraphicBundle::DrawOperation drawOperation);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultFillColor

Sets the default fill color.

<code>static void setDefaultFillColor(const IColor& fillColor);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultFillPattern

Sets the default fill pattern.

<code>static void setDefaultFillPattern(unsigned long fillPattern);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicContext

setDefaultMixMode

Sets the default mix mode for the pen and fill operations.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultMixMode(IGraphicBundle::MixMode mixMode);	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultPatternOrigin

Sets the default pattern origin point.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultPatternOrigin(const IPoint& point);	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultPenColor

Sets the default pen color.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultPenColor(const IColor& penColor);	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultPenEndingStyle

Sets the default pen-ending style. The pen width must be greater than 1 for the pen-ending style to have an effect.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultPenEndingStyle(IGraphicBundle::PenEndingStyle penEndingStyle);	<i>Y</i>	<i>Y</i>	<i>N</i>



This member function has no effect on the pen used to draw graphic objects.

setDefaultPenJoiningStyle

Sets the default pen-joining style. The pen width must be greater than 1 for the pen-joining style to have an effect.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultPenJoiningStyle(IGraphicBundle::PenJoiningStyle penJoiningStyle);	<i>Y</i>	<i>Y</i>	<i>N</i>



This member function has no effect on the pen used to draw graphic objects.

setDefaultPenPattern

Sets the default pen pattern.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultPenPattern(unsigned long penPattern);	<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicContext



This member function has no effect on the pen used to draw graphic objects.

setDefaultPenType

Sets the default pen type. The pen width must be 1 to use a nonsolid pen type.

<pre>static void setDefaultPenType(IGraphicBundle::PenType penType);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

setDefaultPenWidth

Sets the default pen width.

<pre>static void setDefaultPenWidth(unsigned long penWidth);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

Device Space Transformation

The *device space* is the coordinate space in which a picture is drawn before it appears in a display screen window or on the printer or plotter.

The device space is defined in device-specific units. Depending upon the mapping mode used, device coordinate units can be pels, increments of an inch, or increments of a meter, or they can be arbitrary.

The device transformation is defined by two rectangles. The first rectangle is called the "presentation page" in the OS/2 operating system and the "window extent" in Windows. The second is called the "page viewport" in the OS/2 operating system and the "viewport extent" in Windows. The device transformation is established when the graphic context is created or resized.

With the exception of the isotropic mapping mode, the page size and view port rectangle are fixed and are, therefore, predetermined. If you attempt to set the page size or viewport rectangle when the graphic context's mapping mode is not isotropic, the operations are ignored.

pageSize Returns the page size of the graphic context.

<pre>virtual ISize pageSize() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------



This function is equivalent to querying the presentation page size using GpiQueryPS.



The function is equivalent to querying the window extent using GetWindowExtEx.

setPageSize Sets the page size of the graphic context. This function is ignored if the graphic context's mapping mode is not isotropic.

IGraphicContext

<pre>virtual IGraphicContext& setPageSize(const ISize& pageSize);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

PM The function is equivalent to setting the page size when using GpiCreatePS or GpiSetPS.

Win This function is equivalent to setting the window extent using SetWindowExtEx.

setViewportRect

Sets the viewport rectangle of the graphic context. This function is ignored if the graphic context's mapping mode is not isotropic.

<pre>virtual IGraphicContext& setViewportRect(const IRectangle& viewportRectangle);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

PM This function is equivalent to setting the page viewport rectangle using GpiSetPageViewport.

Win This function is equivalent to setting the viewport extent using SetViewportExtEx. The width and height of the argument IRectangle are used to specify the horizontal and vertical extents of the viewport rectangle.

viewportRect Returns the viewport rectangle.

<pre>virtual IRectangle viewportRect() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

PM This function is equivalent to querying the page viewport using GpiQueryPageViewport.

Win The functions is equivalent to querying the viewport extent using GetViewportExtEx.

Drawing

Use these members to render the graphic objects on the device associated with the graphic context. Typically, you use the graphic object's drawOn member to render a graphic object. These functions are provided in this class to consolidate the native drawing operations.

draw Draws the graphic object on the device associated with the graphic context.

1	<pre>virtual IGraphicContext& draw(const IGpie& pie);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

IGraphicContext

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

2	<pre>virtual IGraphicContext& draw(const IGLine& line);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

3	<pre>virtual IGraphicContext& draw(const IGPolyline& polyline);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

4	<pre>virtual IGraphicContext& draw(const IGPolygon& polygon);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

5	<pre>virtual IGraphicContext& draw(const IGEllipse& geometry);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

6	<pre>virtual IGraphicContext& draw(const IRectangle& graphicRectangle);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td>Y</td> <td>Y</td> <td>N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

IGraphicContext

7	<pre>virtual IGraphicContext& draw(const IGArc& arc);</pre>	<table border="0"> <tr> <td style="text-align: center;"><u>Win</u></td> <td style="text-align: center;"><u>PM</u></td> <td style="text-align: center;"><u>Motif</u></td> </tr> <tr> <td style="text-align: center;"><i>Y</i></td> <td style="text-align: center;"><i>Y</i></td> <td style="text-align: center;"><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

8	<pre>virtual IGraphicContext& draw(const IG3PointArc& arc);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

9	virtual IGraphicContext& draw(const IGChord& chord);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	---	-----------------	----------------	-------------------

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

10	<pre>virtual IGraphicContext& draw(const IGString& graphicString);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

11	virtual IGraphicContext& draw(const IGList& list);	Win <i>Y</i>	PM <i>Y</i>	Motif <i>N</i>
-----------	---	------------------------	-----------------------	--------------------------

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

12	<pre>virtual IGraphicContext& draw(const IRegion& region);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

IGraphicContext

Exceptions	
IAccessError	An error occurred while attempting to draw the graphic object. Refer to the exception text for specific error information.

Drawing Attributes

Use these members to set and query the current drawing attributes for the graphic context.

backgroundColor

Returns the current background color.

<pre>virtual IColor backgroundColor() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

backgroundMixMode

Returns the current background mix mode.

<pre>virtual IGraphicBundle::BackgroundMixMode backgroundMixMode() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

drawOperation

Returns the current drawing operation.

<pre>virtual IGraphicBundle::DrawOperation drawOperation() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

fillColor

Returns the current fill color.

<pre>virtual IColor fillColor() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

fillPattern

Returns the current fill pattern.

<pre>virtual unsigned long fillPattern() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

graphicBundle

Queries the current attributes set for the graphic context and returns them as a bundle.

IGraphicContext

virtual IGraphicBundle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
graphicBundle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

mixMode Returns the current pen mix mode.

virtual IGraphicBundle::MixMode	<u>Win</u>	<u>PM</u>	<u>Motif</u>
mixMode() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the mix mode. Refer to the exception text for specific error information.

patternOrigin Returns the current fill pattern origin point. The pattern origin defines a point from which the pen and fill patterns spread horizontally and vertically. The lower-left corner is aligned on this point and is expressed in world coordinate space.

virtual IPoint	<u>Win</u>	<u>PM</u>	<u>Motif</u>
patternOrigin() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pattern origin. Refer to the exception text for specific error information.

penColor Returns the current pen color.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen color. Refer to the exception text for specific error information.

penEndingStyle Returns the current pen-ending style.

virtual IGraphicBundle::PenEndingStyle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
penEndingStyle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen-ending style. Refer to the exception text for specific error information.

IGraphicContext

penJoiningStyle

Returns the current pen-joining style.

```
virtual IGraphicBundle::PenJoiningStyle  
penJoiningStyle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen-joining style. Refer to the exception text for specific error information.

penPattern

Returns the current pen pattern.

```
virtual unsigned long  
penPattern() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen pattern. Refer to the exception text for specific error information.

penType

Returns the current pen type.

```
virtual IGraphicBundle::PenType  
penType() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen type. Refer to the exception text for specific error information.

penWidth

Returns the current pen width.

```
virtual unsigned long  
penWidth() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the pen width. Refer to the exception text for specific error information.

setBackgroundColors

Sets the current background color.

IGraphicContext

```
virtual IGraphicContext&  
    setBackgroundColor( const IColor& backgroundColor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the background color. Refer to the exception text for specific error information.

setBackgroundMixMode

Sets the current background mix mode.

```
virtual IGraphicContext&  
    setBackgroundMixMode(  
        IGraphicBundle::BackgroundMixMode backgroundMixMode);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the background mix mode. Refer to the exception text for specific error information.

setDrawOperation

Sets the current draw operation for closed figures (frame, fill, or fill and frame).

```
virtual IGraphicContext&  
    setDrawOperation(  
        IGraphicBundle::DrawOperation drawOperation);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set draw operation. Refer to the exception text for specific error information.

setFillColor

Sets the current fill color.

```
virtual IGraphicContext&  
    setFillColor( const IColor& fillColor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the fill color. Refer to the exception text for specific error information.

setFillPattern

Sets the current fill pattern.

```
virtual IGraphicContext&  
    setFillPattern( unsigned long fillPattern );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicContext

Exceptions	
IAccessError	An error occurred while attempting to set the fill pattern. Refer to the exception text for specific error information.

setGraphicBundle

Sets the drawing attributes for the graphic context to those you set in the graphic bundle.

```
virtual IGraphicContext&  
    setGraphicBundle( const IGraphicBundle& graphicBundle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setMixMode Sets the current mix mode.

```
virtual IGraphicContext&  
    setMixMode( IGraphicBundle::MixMode mixMode );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the mix mode. Refer to the exception text for specific error information.

setPatternOrigin

Sets the current fill pattern origin point.

```
virtual IGraphicContext&  
    setPatternOrigin( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the pattern origin. Refer to the exception text for specific error information.

setPenColor Sets the current pen color.

```
virtual IGraphicContext&  
    setPenColor( const IColor& penColor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the pen color. Refer to the exception text for specific error information.

IGraphicContext

setPenEndingStyle

Sets the current pen-ending style. The pen width must be greater than 1 for the pen-ending style to have an effect.

```
virtual IGraphicContext&
    setPenEndingStyle(
        IGraphicBundle::PenEndingStyle penEndingStyle);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The pen-ending style is not currently supported on Windows 95.



This member function has no effect on the pen used to draw graphic objects.

Exceptions	
IAccessError	An error occurred while attempting to set the pen-ending style. Refer to the exception text for specific error information.

setPenJoiningStyle

Sets the current pen-joining style. The pen width must be greater than 1 for the pen-joining style to have an effect.

```
virtual IGraphicContext&
    setPenJoiningStyle(
        IGraphicBundle::PenJoiningStyle penJoiningStyle);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The pen-joining style is not currently supported on Windows 95.



This member function has no effect on the pen used to draw graphic objects.

Exceptions	
IAccessError	An error occurred while attempting to set the pen-joining style. Refer to the exception text for specific error information.

setPenPattern

Sets the current pen pattern.

```
virtual IGraphicContext&
    setPenPattern( unsigned long penPattern );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This member function has no effect on the pen used to draw graphic objects.

Exceptions	
IAccessError	An error occurred while attempting to set the pen pattern. Refer to the exception text for specific error information.

setPenType

Sets the current pen type. The pen width must be 1 for a pen type other than solid to be used.

IGraphicContext

```
virtual IGraphicContext&
    setPenType( IGraphicBundle::PenType penType );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



Only the solid and invisible pen types are currently supported on Windows 95.



The IGraphicBundle::doubleDot type is replaced by IGraphicBundle::dot on pens used to draw graphic objects. The IGraphicBundle::alternate type is replaced by IGraphicBundle::solid on pens used to draw graphic objects.

Exceptions	
IAccessError	An error occurred while attempting to set the pen type. Refer to the exception text for specific error information.

setPenWidth Sets the current pen width.

```
virtual IGraphicContext&
    setPenWidth( unsigned long width );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



Only the IGraphicBundle::solid type makes use of pen widths other than 1.

Exceptions	
IAccessError	An error occurred while attempting to set the pen width. Refer to the exception text for specific error information.

Font Operations

Use these members to set and query information for the font that is used by the graphic context when drawing text.

currentFont Returns the font currently set in the graphic context.

```
virtual IFont
    currentFont() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

hasFont Returns true if a font has been set for the graphic context. If the default font is being used, the function returns false.

```
virtual IBase::Boolean
    hasFont() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

resetFont Sets the graphic context's current font to the system default font.

```
virtual IGraphicContext&
    resetFont();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IGraphicContext

Exceptions	
IAccessError	An error occurred while attempting to reset the font. Refer to the exception text for specific error information.

setFont Sets the graphic context's current font to the font specified.

```
virtual IGraphicContext&
    setFont( const IFont& font );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	An error occurred while attempting to set the font. Refer to the exception text for specific error information.

Hit Testing

Hit testing is the process of determining which graphic objects, if any, are contained within an area of interest. When you select an area of interest on the screen, you usually move the pointer to the applicable point and signal (by clicking the mouse, for example) that this is the graphic object you want. The *area of interest* is a small rectangle centered at the hit point; its size is determined by the hit aperture size. The hit testing process draws graphic objects without rendering them on the screen.

Use these members to provide hit testing support for graphic objects in other classes. Normally, you do not need to call these functions to perform hit testing. Instead, use IGraphic::contains, or IGList::topGraphicUnderPoint, IGList::bottomGraphicUnderPoint, or use the IGList hit cursor to perform hit testing.

hitApertureSize

Returns the current hit aperture size.

```
virtual ISize
    hitApertureSize() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

hitPoint Returns the current hit point.

```
virtual IPoint
    hitPoint() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

isHitTesting This member function returns true if a hit testing is in progress; otherwise it returns false.

IGraphicContext

```
IBase::Boolean  
isHitTesting() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



This function is useful when you are creating a class which derives from IGraphic. In your classes drawOn member function you should call this function to determine if the graphic context is hit testing. If this function returns true, you should determine if your graphic object intersects with the hit aperture. The hit aperture is determined by creating a rectangle the size of the hit aperture which is centered at the hit point. You should call IGraphic::setHitSelected(true) if your graphic object intersects the hit rectangle. If this function returns false, you should draw your graphic object.

setHitApertureSize

Sets the size of the area around the hit point to use for determining hit tests.

```
virtual IGraphicContext&  
setHitApertureSize( const ISize& hitSize );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the hit aperture size. Refer to the exception text for specific error information.

setHitPoint

Sets the current hit point. Typically, this hit point is the point generated when the user presses a mouse button.

```
virtual IGraphicContext&  
setHitPoint( const IPoint& hitPoint );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the hit point. Refer to the exception text for specific error information.

startHitTesting

Starts the hit-testing process. Any graphic drawn using this graphic context after this function has been called sets its selection state to true if the current hit point is contained within the graphic. Draw functions called after this function call are not visible.

```
virtual IGraphicContext&  
startHitTesting();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to start hit-testing. Refer to the exception text for specific error information.

IGraphicContext

stopHitTesting

Stops the hit-testing process. All subsequent draw functions are visible.

```
virtual IGraphicContext&
    stopHitTesting();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to stop hit-testing. Refer to the exception text for specific error information.

Mapping Modes

Use these members to define the unit of measure used to map between the page space and the device space. The supported mapping modes are as follows:

pels	Pel coordinates. This is the default when the graphic context is created.
lowMetric	Units of 0.1 millimeter.
highMetric	Units of 0.01 millimeter.
lowEnglish	Units of 0.01 inch.
highEnglish	Units of 0.001 inch.
twips	Units of 1/1440 inch.
isotropic	Logical units in the X-direction and Y-direction are equal when mapping between page space and device space. The aspect ratio is preserved.

mappingMode

Retrieves the units defined for the graphic context.

```
virtual MappingMode
    mappingMode() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the mapping mode. Refer to the exception text for specific error information.

setMappingMode

Sets the units for the graphic context.

```
virtual IGraphicContext&
    setMappingMode( MappingMode mappingMode = pels );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the mapping mode. Refer to the exception text for specific error information.

IGraphicContext

Recoordination

Recoordination is the process of mapping from one coordinate system to another. This is useful if you are creating an application that you plan to provide on multiple platforms. For example, you may have initially created an application on OS/2 that draws graphic objects using coordinates relative to the lower-left corner of a window. When you move the application to Windows, by default the graphic objects are now drawn relative to the upper-left corner of a window. By setting the recoordination height of the graphic context to the height of the window (or device) you are drawing into, graphic objects are then drawn relative to a different corner of the window. In the case of Windows and Motif, recoordination means graphic objects are drawn relative to the lower left corner of a window. In the case of OS/2, *recoordination* means graphic objects are drawn relative to the upper left corner of a window.

Recoordination is done only when the application orientation (window origin) differs from the native application orientation and you have set a nonzero recoordination height.

Recoordination of graphic objects is not supported if you set a transform on a graphic object.

recoordinationHeight

Returns the current recoordination height. The default recoordination height is zero.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
recoordinationHeight() const;	Y	Y	N

setRecoordinationHeight

Set this value to the height of the window (or device) in which you are drawing graphic objects. This function should be called each time the height of the window you are drawing into changes. By setting the recoordination height to zero, the graphic context will no longer recoordinate graphic objects regardless of the current application orientation.

IGraphicContext&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setRecoordinationHeight(unsigned long height);	Y	Y	N

Region Operations

Use these members to query and set a graphic context's clip region. A *region* is an area in device space that is defined by one or more rectangular areas. You can use a region to determine the clip area used when drawing on a device. When you use a region as a clip region, the areas that are included in a region allow graphics that are contained or that intersect the region to be output to the device. Areas outside of the clip region are not output to the device.

IGraphicContext

clearClipRegion

Clears the current clip region.

```
virtual IGraphicContext&
    clearClipRegion();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to clear the clip region. Make sure you are not simultaneously using the graphic context in another thread.

clipRegion

Returns a handle to the current clip region. If you have not set a region as the clip region, then a null handle is returned.

```
virtual IRegionHandle
    clipRegion() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setClipRegion

Sets the clip region for the graphic context.

```
1 virtual IGraphicContext&
    setClipRegion( const IRegionHandle& region );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Sets the argument region as the new clip region for the graphic context.

Exceptions	
IAccessError	An error occurred while attempting to set the clipping region. Refer to the exception text for specific error information.

```
2 virtual IGraphicContext&
    setClipRegion( const IRegion& region );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Sets the argument region as the new clip region for the graphic context.

Exceptions	
IAccessError	An error occurred while attempting to set the clipping region. Refer to the exception text for specific error information.

World Space Transformation

Use these members to set and query the current world coordinate space transform matrix.

IGraphicContext

setWorldTransformMatrix

Sets the world coordinate space transform matrix using the specified transform method.

```
virtual IGraphicContext&
    setWorldTransformMatrix(
        const ITransformMatrix& transformMatrix,
        IGraphic::TransformMethod method = IGraphic::replace );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to set the world coordinate space transform matrix. Refer to the exception text for specific error information.

worldTransformMatrix

Returns the current world coordinate space transform matrix.

```
virtual ITransformMatrix
    worldTransformMatrix() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to query the world coordinate space transform matrix. Refer to the exception text for specific error information.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

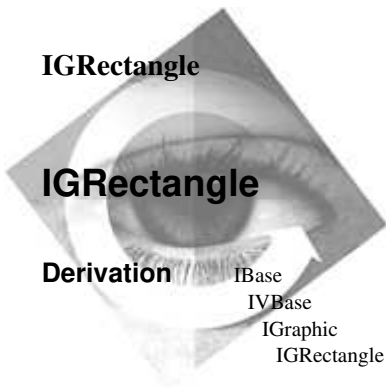
IBase		
recoverable	unrecoverable	

IGraphicContext

MappingMode

```
MappingMode {  
    pels,          lowMetric,  highMetric, lowEnglish, highEnglish,  
    twips,         isotropic  
};
```

The mapping mode defines the unit of measure used to map between the page space and the device space.



Inherited By None.

Header File igrect.hpp

Members				
Member	Page	Member	Page	
Constructor	407	operator ==	407	
drawOn	408	rounding	408	
enclosingRect	408	setEnclosingRect	409	
operator !=	407	setRounding	408	
operator =	407	~IGRectangle	408	

The IRectangle class draws two-dimensional rectangles. An IGRectangle may be filled, framed, or filled and framed.

You can optionally round the corners of the rectangle by specifying the full length of the horizontal and vertical axes of an ellipse. The corners of the rectangle are rounded by a quarter of the ellipse.

When you draw an IGRectangle, the following graphic bundle drawing attributes affect its appearance:

- Draw operation
- Pen color
- Fill color
- Background color
- Mix mode
- Background mix mode
- Pen width
- Pen type
- Pen-joining style
- Pen pattern
- Fill pattern
- Pattern origin

Public Functions

Comparisons

Use these members to compare two IGRectangle objects.

operator != Returns true if the rectangles are not identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const IGRectangle& gRectangle) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

operator == Returns true if the rectangles are identical (includes the graphic bundle attributes and the transform matrix).

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGRectangle& gRectangle) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGRectangle

1	IGRectangle(const IRectangle& rectangle = IRectangle ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGRectangle object from an IRectangle object. The position and size of the IGRectangle is set by the argument IRectangle.

rectangle An IRectangle object specifying size and position. (Optional).

2	IGRectangle(const IGRectangle& gRectangle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGRectangle object from another IGRectangle object.

gRectangle A reference to an IGRectangle object.

operator = Use this function to assign a IGRectangle object to another IGRectangle object.

IGRectangle&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IGRectangle& gRectangle);	<i>Y</i>	<i>Y</i>	<i>N</i>

IGRectangle

~IGRectangle

<code>virtual ~IGRectangle();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Corner Rounding

Use these members to round the corners of the IGRectangle object. You control the corner rounding by specifying the full length of the horizontal and vertical axes of an ellipse. The corners of the box are rounded by a quarter of the ellipse. If the horizontal and vertical axes are equal, the rectangle is rounded by using a quarter circle.

rounding Returns the full length of the horizontal and vertical axes used to determine corner rounding. By default, the corners of the rectangle are not rounded.

<code>virtual IPair rounding() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setRounding Sets the full length of the horizontal and vertical axes of the ellipse used to determine corner rounding.

<code>virtual IGRectangle& setRounding(const IPair& ellipseAxis = IPair (0 , 0));</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Drawing

Use these members to render an IGRectangle object on a device.

drawOn Draws the rectangle on the device associated with the graphic context.

<code>virtual IGRectangle& drawOn(IGraphicContext& graphicContext);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Enclosing Rectangle

Use these members to define the size and position of the IGRectangle object.

enclosingRect

Returns the dimensions of the IGRectangle.

<code>virtual IRectangle enclosingRect() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IGRectangle

setEnclosingRect

Sets the dimensions of the IGRectangle.

```
virtual IGRectangle&
    setEnclosingRect( const IRectangle& rectangle );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File igregion.hpp

Members				
Member	Page	Member	Page	
Constructor	411	operator -=	418	
clear	413	operator =	412	
drawOn	413	operator const IRegionHandle	412	
moveBy	421	operator ^=	419	
operator &=	415	~IGRegion	412	
operator +=	417			

The IGRRegion class is a graphic object that can be composed of one or more closed figures. You can use an IGRRegion to construct a shape from one or more closed figures. You can draw the region on a graphic context or use it as a clip region when drawing other graphic objects on a graphic context.

When you draw an IGRRegion, the following graphic bundle drawing attributes affect a region's appearance:

- Pen color
- Mix mode
- Pen pattern
- Pattern origin

The coordinates you use to define a region are specified in device space. For this reason the world transform functions that are declared in IGraphic have no effect and are overridden as private functions in IGRRegion.

Constructing an IGRRegion is a process that simply involves adding, subtracting, logical XORing, or logical ANDing a closed figure or another IGRRegion.

Public Functions

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGRegion

1	IGRegion();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Use the default constructor to create an IGRRegion object. The region is initially empty.

Exceptions	
IAccessError	An error occurred while attempting to create the region. Refer to the exception text for specific error information.

2	<code>IGRegion(const IRegionHandle& regionHandle);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGRRegion object from an IRegionHandle object. The IGRRegion is initialized to the region defined by the region handle.

<i>regionHandle</i>	A handle to an existing region.
---------------------	---------------------------------

Exceptions	
IAccessError	An error occurred while attempting to create the region. Refer to the exception text for specific error information.

3	<code>IGRegion(const IGRegion& region);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an `IGRegion` object from another `IGRegion` object.

<i>region</i>	A reference to an IGRRegion object.
---------------	-------------------------------------

Exceptions	
IAccessError	An error occurred while attempting to create the region. Refer to the exception text for specific error information.

4	<pre>IGRegion(const IGraphicContext& graphicContext, const IRectangle& rectangle);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td>Y</td> <td>Y</td> <td>N</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

IGRegion

Use this function to create an IRegion object from an IGraphicContext object and an IRectangle object. The region is created from the area specified by the rectangle in device coordinates for the argument graphic context.

<i>graphicContext</i>	An IGraphicContext object.
<i>rectangle</i>	Rectangular area in device coordinates.

Exceptions	
IAccessError	An error occurred while attempting to create the region. Refer to the exception text for specific error information.

5	<code>IGRegion(const IWindowHandle& windowHandle);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an `IGRegion` object from an `IWindowHandle` object. The region is created from the windows update region.

<i>windowHandle</i>	Window handle from which the update region is created.
---------------------	--

Exceptions	
IAccessError	An error occurred while attempting to create the region. Refer to the exception text for specific error information.

operator = Use this function to construct an IGRRegion object from another IGRRegion object.

IGRegion& operator =(const IGRegion& region);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

~IGRegion

<code>virtual</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IGRegion();</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

Conversion

Use these members to access the `IGRegion` as an `IRegionHandle` object.

operator const IRegionHandle

Returns a handle to the IGRRegion.

operator const IRegionHandle() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IGRegion

Drawing

Use these members to render an IGRegion object on a device.

drawOn Draws the region on the device associated with the graphic context.

<pre>virtual IGRegion& drawOn(IGraphicContext& graphicContext);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	An error occurred while attempting to draw the region. Refer to the exception text for specific error information.

Modifying a Region

Use these members to modify the area of a region. See Figure 6 for various ways to combine overlapping regions. See Figure 7 for various ways to combine disjoining regions.

clear Sets the region to a null region.

<pre>virtual IGRegion& clear();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Operators

Use these members to combine an object with a region.

IGRegion

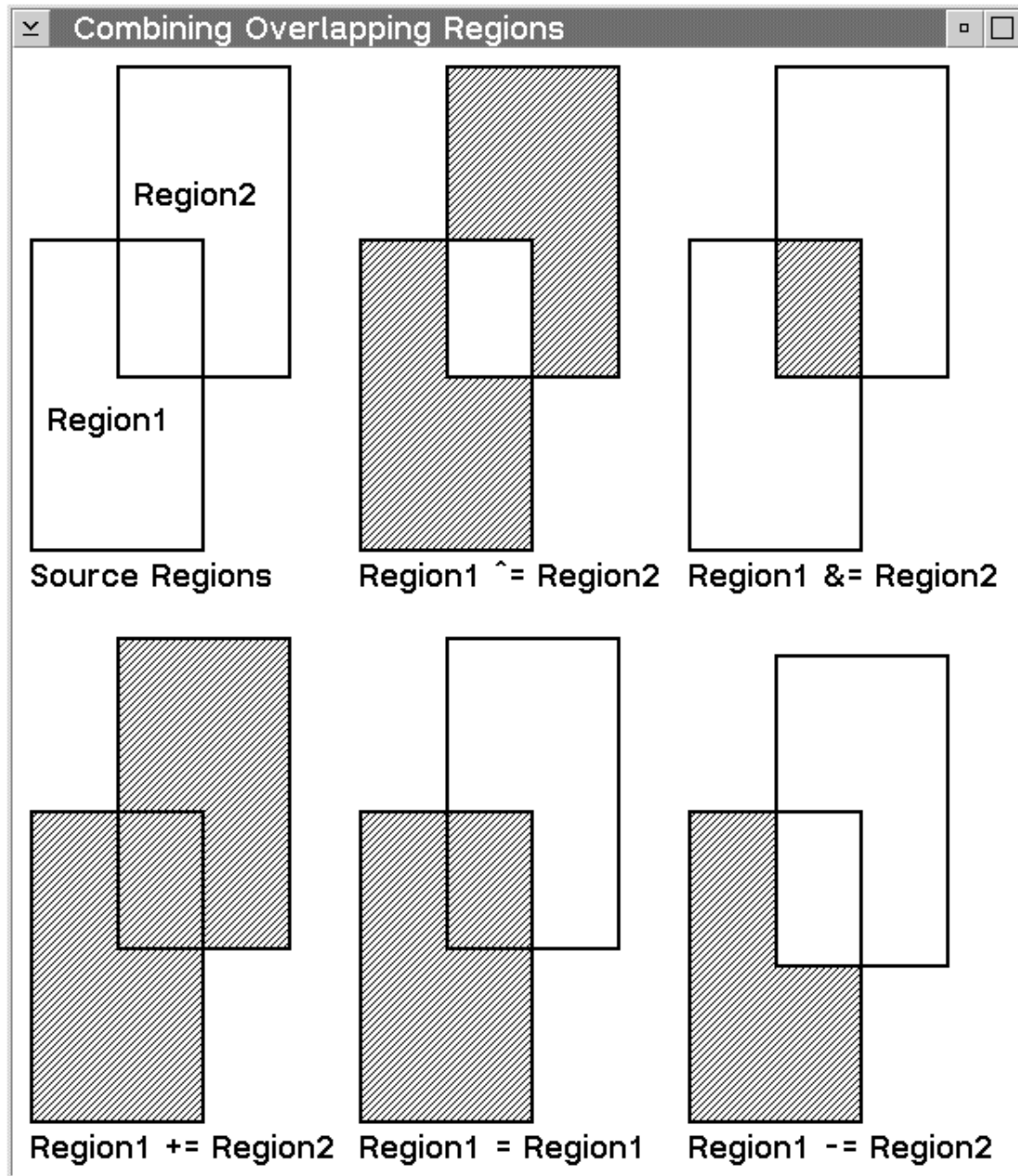


Figure 6. Combining Overlapping Regions

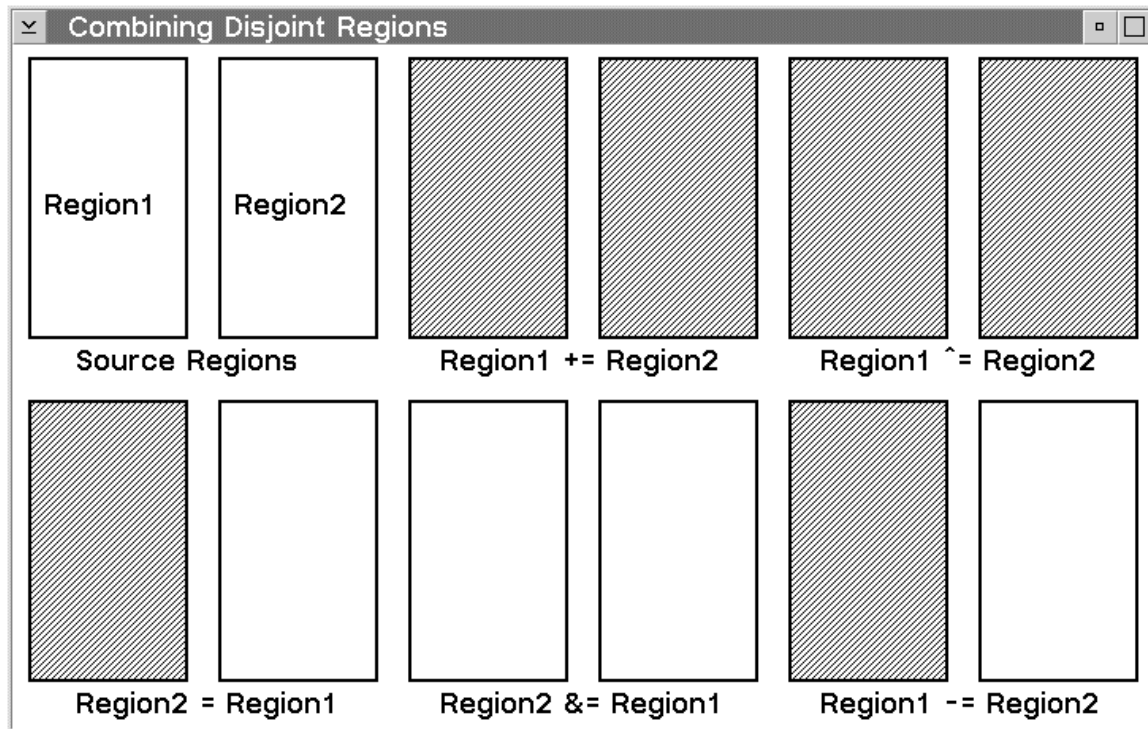


Figure 7. Combining Disjoint Regions

operator &=

1 IGRegion& operator &= (const IGRegion& region); Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the two regions.

Exceptions	
IAccessError	An error occurred while attempting to combine the regions. Refer to the exception text for specific error information.

2 IGRegion& operator &= (const IGRectangle& rect); Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the region and the area of the rectangle.

IGRegion

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

3 IGRegion&
operator &=(const IGPie& pie);
Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the region and the area of the pie.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

4 IGRegion&
operator &=(const IGChord& chord);
Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the region and the area of the chord.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

5 IGRegion&
operator &=(const IGPolygon& polygon);
Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the region and the area of the polygon.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

6 IGRegion&
operator &=(const IGEllipse& ellipse);
Win PM Motif
Y Y N

Sets the region to the area defined by the intersection of the region and the area of the ellipse.

IGRegion

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

operator +=

- | | | | | |
|----------|---|------------------------|-----------------------|--------------------------|
| 1 | IGRegion&
operator +=(const IGPie& pie); | Win
<i>Y</i> | PM
<i>Y</i> | Motif
<i>N</i> |
|----------|---|------------------------|-----------------------|--------------------------|

Sets the region to the area defined by the union of the region and the area of the pie.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

- | | | | | | | | | |
|-------------------|--|---|-------------------|------------------|---------------------|----------|----------|----------|
| 2 | <pre>IGRegion& operator +=(const IGRectangle& rect);</pre> | <table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table> | <u>Win</u> | <u>PM</u> | <u>Motif</u> | <i>Y</i> | <i>Y</i> | <i>N</i> |
| <u>Win</u> | <u>PM</u> | <u>Motif</u> | | | | | | |
| <i>Y</i> | <i>Y</i> | <i>N</i> | | | | | | |

Sets the region to the area defined by the union of the region and the area of the rectangle.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

- | | | | | | | | | |
|-------------------|---|---|-------------------|------------------|---------------------|----------|----------|----------|
| 3 | <pre>IGRegion& operator +=(const IGChord& chord);</pre> | <table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table> | <u>Win</u> | <u>PM</u> | <u>Motif</u> | <i>Y</i> | <i>Y</i> | <i>N</i> |
| <u>Win</u> | <u>PM</u> | <u>Motif</u> | | | | | | |
| <i>Y</i> | <i>Y</i> | <i>N</i> | | | | | | |

Sets the region to the area defined by the union of the region and the area of the chord.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

- | | | | | | | | | |
|-------------------|---|---|-------------------|------------------|---------------------|----------|----------|----------|
| 4 | <pre>IGRegion& operator +=(const IGPolygon& polygon);</pre> | <table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table> | <u>Win</u> | <u>PM</u> | <u>Motif</u> | <i>Y</i> | <i>Y</i> | <i>N</i> |
| <u>Win</u> | <u>PM</u> | <u>Motif</u> | | | | | | |
| <i>Y</i> | <i>Y</i> | <i>N</i> | | | | | | |

Sets the region to the area defined by the union of the region and the area of the polygon.

IGRegion

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

5 `IGRegion& operator +=(const IGEllipse& ellipse);` Win PM Motif
Y Y N

Sets the region to the area defined by the union of the region and the area of the ellipse.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

6 `IGRegion& operator +=(const IGRegion& region);` Win PM Motif
Y Y N

Sets the region to the area defined by the union of the two regions.

Exceptions	
IAccessError	An error occurred while attempting to combine the regions. Refer to the exception text for specific error information.

operator -=

1 `IGRegion& operator -=(const IGChord& chord);` Win PM Motif
Y Y N

Sets the region to the area defined by the region minus the area of the chord.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

2 `IGRegion& operator -=(const IGRectangle& rect);` Win PM Motif
Y Y N

Sets the region to the area defined by the region minus the area of the rectangle.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

IGRegion

3	<pre>IGRegion& operator --(const IGPie& pie);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Sets the region to the area defined by the region minus the area of the pie.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

4	<pre>IGRegion& operator --(const IGPolygon& polygon);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Sets the region to the area defined by the region minus the area of the polygon.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

5	<pre>IGRegion& operator ==(const IGEllipse& ellipse);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Sets the region to the area defined by the region minus the area of the ellipse.

Exceptions	
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.

6	<pre>IGRegion& operator ==(const IGRegion& region);</pre>	<table border="0"> <tr> <td style="text-align: right;"><u>Win</u></td> <td style="text-align: right;"><u>PM</u></td> <td style="text-align: right;"><u>Motif</u></td> </tr> <tr> <td style="text-align: right;"><i>Y</i></td> <td style="text-align: right;"><i>Y</i></td> <td style="text-align: right;"><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Sets the region to the area defined by the target region minus the area of the operand region.

Exceptions	
IAccessError	An error occurred while attempting to combine the regions. Refer to the exception text for specific error information.

operator ^=

1	IGRegion&	Win	PM	Motif
	operator ^=(const IGRectangle& rect);	Y	Y	N

Sets the region to the area defined by the logical xor of the region and the rectangle.

IGRegion

Exceptions		
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.	

2 IGRegion&
operator ^=(const IGPie& pie);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Sets the region to the area defined by the logical xor of the region and the pie.

Exceptions		
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.	

3 IGRegion&
operator ^=(const IGChord& chord);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Sets the region to the area defined by the logical xor of the region and the chord.

Exceptions		
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.	

4 IGRegion&
operator ^=(const IGPolygon& polygon);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Sets the region to the area defined by the logical xor of the region and the polygon.

Exceptions		
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.	

5 IGRegion&
operator ^=(const IGEllipse& ellipse);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Sets the region to the area defined by the logical xor of the region and the ellipse.

Exceptions		
IAccessError	An error occurred while attempting to combine the area with the region. Refer to the exception text for specific error information.	

6 IGRegion&
operator ^=(const IGRegion& region);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IGRegion

Sets the region to the area defined by the logical xor of the two regions.

Exceptions		
IAccessError		An error occurred while attempting to combine the regions. Refer to the exception text for specific error information.

Positioning

Use these members to move an IGRegion object in device space.

moveBy Moves the region by the amount specified by the argument point.

```
virtual IGRegion&
moveBy( const IPoint& point );
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
IAccessError		An error occurred while attempting to move the region. Refer to the exception text for specific error information.

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IGRegion

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IGString

IGString

Derivation IBase
IVBase
IGraphic
IGString

Inherited By None.

Header File igstring.hpp

Members	Member	Page	Member	Page
	Constructor	425	operator ==	424
	clippingRect	424	position	427
	drawOn	426	removeFont	426
	font	426	resetClippingRect	424
	hasFont	426	setClippingRect	424
	isClippingRectSet	424	setFont	426
	moveTo	427	setText	427
	operator !=	424	text	427
	operator =	425	~IGString	426

The IGString class is a graphic object class that allows you to draw text. An IGString object can optionally have a font associated with it that is used when you draw the IGString object. If you do not supply a font object for the IGString object, the graphic context current font is used when you draw the IGString object.

In all the constructors you provide a location point of where the text drawing starts. Text alignment and font direction determine where the text is positioned in relation to the location point you specify.

When you draw a IGString object, the following graphic bundle drawing attributes affect its appearance:

- Pen color
- Background color
- Mix mode
- Background mix mode

Public Functions

IGString

Clipping Rectangle

Use these members to set, query, and remove the clipping rectangle used when drawing the text. If a clipping rectangle is set, the text is clipped to the boundary to the rectangle.

clippingRect Returns the clipping rectangle. If a clipping rectangle has not been set, a default rectangle (0,0,0,0) is returned.

<pre>virtual IRectangle clippingRect() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

isClippingRectSet

Returns true if a clipping rectangle is set.

<pre>virtual IBase::Boolean isClippingRectSet() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

resetClippingRect

Removes the clipping rectangle.

<pre>virtual IGString& resetClippingRect();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setClippingRect

Sets the clipping rectangle used when drawing the text.

<pre>virtual IGString& setClippingRect(const IRectangle& rectangle);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Comparisons

Use these members to compare two IGString objects.

operator != Returns true if the text objects are not identical (includes location, font, the graphic bundle attributes, and the transform matrix).

<pre>IBase::Boolean operator !=(const IGString& graphicString) const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

operator == Returns true if the text objects are identical (includes location, font, the graphic bundle attributes and the transform matrix).

IGString

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IGString& graphicString) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, destruct, copy, and assign objects of this class.

IGString

1	IGString(const IString& string, const IPoint& point, const IFont& font);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGString object from an IString, an IPoint object specifying where to draw the string, and an IFont object specifying the font to use when you draw the IGString object.

<i>string</i>	Characters to be drawn.
<i>point</i>	Location of where to draw the characters.
<i>font</i>	Font used when drawing the characters. (Optional)

2	IGString(const IString& string, const IPoint& point = IPoint ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGString object from an IString object and optionally an IPoint object specifying where to draw the string.

<i>string</i>	Characters to be drawn.
<i>point</i>	Location of where to draw the characters.

3	IGString(const IGString& graphicString);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to construct an IGString object from another IGString object.

<i>graphicString</i>	A reference to an IGString object.
----------------------	------------------------------------

operator = Use this function to assign a IGString object to another IGString object.

IGString&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator =(const IGString& graphicString);	<i>Y</i>	<i>Y</i>	<i>N</i>

IGString

~IGString

<pre>virtual ~IGString();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Drawing

Use these members to render an IGString object on a device.

drawOn Draws the string at the current position. The string is drawn using the object's font if set. If you have not specified a font for this object (either in the constructor or via the setFont function), then the text is drawn using the graphic context's current font.

<pre>virtual IGString& drawOn(IGraphicContext& graphicContext);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Font

Use these members to set, query, and remove the font used when you draw an IGString object. If you have not set a font for an IGString object, the graphic context's current font is used when you draw an IGString object.

font Returns the current font being used to draw the text. If you did not set a font on the object or constructed the object with a font object, this function returns the system default font.

<pre>virtual IFont font() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

hasFont Returns true if a font was specified in the constructor or set using setFont function.

<pre>virtual IBase::Boolean hasFont() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

removeFont Removes the font object from the graphic text. The font is deleted.

<pre>virtual IGString& removeFont();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setFont Sets the font to be used when drawing the text.

<pre>virtual IGString& setFont(const IFont& font);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IGString

Text

Use these members to set and query the text of an IGString object.

setText Sets the graphic object's text.

1	<code>virtual IGString& setText(const char* text);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

2	<code>virtual IGString& setText(const IResourceId& text);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

text Returns the graphic object's text.

<code>virtual IString text() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Text Positioning

Use these members to set and query the position of the text.

moveTo Sets the starting location of where the text is to be drawn.

<code>virtual IGString& moveTo(const IPoint& point);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

position Returns the starting location of where the text is to be drawn.

<code>virtual IPoint position() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IGraphic		
boundingRect	isHitSelectable	setHitSelectable
contains	isHitSelected	setHitSelected
drawOn	removeGraphicBundle	setId
graphicBundle	resetTransformMatrix	setTransformMatrix
hasGraphicBundle	rotateBy	setTransformMethod
hasTransformMatrix	scaleBy	transformMatrix

IGString

IGraphic		
id	setGraphicBundle	transformMethod

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IGraphic		
operator !=	operator =	operator ==

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IGUIBundle

Derivation Inherits from none.

Inherited By None.

Header File iguibndl.hpp

Members				
Member	Page	Member	Page	
Constructor	434	handleFileOpen	432	
adoptComponent	429	handleFileSave	433	
adoptFrameWindow	430	handleFileSaveAs	433	
adoptView	430	handleInsertObject	433	
component	435	handlePaste	433	
frameWindow	435	handlePasteLink	433	
handleConvert	430	handlePasteSpecial	434	
handleCopy	431	handleSelectAll	434	
handleCut	431	objectMenuBar	435	
handleDefaultMenuCommands	431	objectToolBar	435	
handleDelete	431	objectView	435	
handleDoVerb	432	okToDiscardFile	436	
handleEditLinks	432	view	435	
handleExit	432	~IGUIBundle	434	
handleFileNew	432			

The GUI bundle is the object that coordinates the component, the model, the view, and the frame window. The default implementation should be satisfactory in most cases, however, you may derive from this class and override functions as required. If you do use a derived class, remember to override `IComponentStationery::createBundle` (p. 15).

Currently, there is only one GUI bundle per application which is fine for single-use servers. Multiple-use servers will be supported in the future.

Public Functions

Adoption

The framework uses these members to adopt items into the bundle. Once adopted, the GUI bundle assumes responsibility for the destruction of the item.

adoptComponent

Adopts a component into the bundle.

IGUIBundle

<code>virtual Boolean adoptComponent(IComponent*);</code>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

*IComponent**
A pointer to the component to be adopted.

adoptFrameWindow

Adopts a frame window into the bundle (taking on the responsibility for deleting the memory), and creates and starts the menu handlers for the window (see the Command Handler functions). The adopted frame window is also known as the out-of-place frame window.

<code>virtual Boolean adoptFrameWindow(IComponentFrameWindow*);</code>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

*IComponentFrameWindow**
A pointer to the frame window to be adopted.

adoptView

Adopts a view into the bundle. The view becomes the client area of the (previously adopted) frame window, and is initialized. The framework uses this view when the application runs stand-alone, or is embedded out-of-place.

<code>virtual Boolean adoptView(IView*);</code>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

*IView** A pointer to the view to be adopted.

Command Handlers

Use these members to handle commands passed in through the user interface.

handleConvert

Called when the end-user selects **Convert** from the **Edit ► Object** menu. This menu option is available only on container components. The default implementation does nothing. Override this function in a derived class to provide conversion functionality.

IGUIBundle

<pre>virtual void handleConvert();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleCopy Called when the end-user selects **Edit ► Copy** or **Edit ► Cut**. It saves a copy of the currently selected item to a temporary file, and puts the file and its description onto the clipboard. When the end-user calls for a paste from the clipboard, the saved items are copied into the target application.

<pre>virtual void handleCopy();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleCut Called when the end-user selects **Edit ► Cut**. This function calls handleCopy. If this is a container, and a single embedded component is selected, then the embedded component is deleted. If **Select All** is checked, the the equivalent of a **File ► New** is executed..

<pre>virtual void handleCut();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleDefaultMenuCommands

The framework calls this function whenever the end-user selects a menu option or toolbar button. The function then calls out to the appropriate handle function (for example, if the end-user selects **File ► New**, this function calls out to handleFileNew).

It is not normally necessary to override this function. Instead, you can place your own command handler before this one, and return false from commands not handled. See the description of ICommandHandler (Vol. II) for more information.

The framework automatically propagates unhandled events up the chain. If you do override this function you should call the base class implementation.

<pre>virtual Boolean handleDefaultMenuCommands(ICommandEvent&);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

ICommandEvent&
A command event.

handleDelete Called when the end-user selects Delete from the **Edit** menu. The default implementation does nothing for non-containers. For containers, if a single embedded component is selected, then the embedded component is deleted. If **Select All** is checked, then all embedded components are deleted.

IGUIBundle

<pre>virtual void handleDelete();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleDoVerb

Called when the end-user selects an operation (verb) from the object's cascade menu. This menu option is available only on container components. Activates the server and passes it the corresponding verb.

<pre>virtual void handleDoVerb(unsigned iVerb);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

The parameter for this function is as follows:

iVerb An OLE operation (verb).

handleEditLinks

Called when the end-user selects **Edit ► Links**. This menu option is available only on container components.

Displays the **Edit Links** dialog, which gives the user several link management options.

<pre>virtual void handleEditLinks();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleExit

Called when the end-user selects **File ► Exit**. Starts a shutdown by prompting the user to save unsaved data, then closes the view, frame window and so forth.

<pre>virtual void handleExit(Boolean close = false);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleFileNew

Called when the end-user selects **File ► New**. Prompts the end-user to save unsaved data, then calls on the component to load an empty storage.

<pre>virtual void handleFileNew();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

handleFileOpen

Called when the end-user selects **File ► Open**. Prompts the end-user to save unsaved data (if any), shows a file open dialog, and loads the selected file.

IGUIBundle

virtual void handleFileOpen();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

handleFileSave

Called when the end-user selects **File ► Save**. Calls on the component to save (that is, stream-out) the data. If there is no file name associated with the component, then this command acts as a handleFileSaveAs.

virtual void handleFileSave();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

handleFileSaveAs

Called when the end-user selects **File ► Save As**. Shows a **Save As** dialog box, then calls on the component to create the file, and stream-out the model.

virtual void handleFileSaveAs(Boolean copy = false);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

handleInsertObject

Called when the end-user selects **Edit ► Insert Object**. This menu option is available only on container components.

Shows the Insert Object dialog, then inserts the selected object (either a new object or a file). New objects are launched in-place if possible, otherwise out-of-place.

virtual void handleInsertObject();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

handlePaste

Called when the end-user selects **Edit►Paste**. Selects the object with the closest matching format currently on the clipboard, and pastes the object into the application as an embedded component.

virtual void handlePaste();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

handlePasteLink

Called when the end-user selects **Edit►Paste Link**. This menu option is available only on container components.

IGUIBundle

Selects the object with the closest matching format currently on the clipboard, and pastes a link to the object into the application.

virtual void handlePasteLink();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
------------------------------------	-----------------	----------------	-------------------

handlePasteSpecial

Called when the end-user selects **Edit►Paste Special**. Only container components have this menu option.

Shows the **Paste Special** dialog, which presents the end-user with three options: **Paste**, **Paste Link**, and **Cancel**. Once the end-user selects an option, the function proceeds accordingly.

virtual void handlePasteSpecial();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

handleSelectAll

Called when the end-user selects **Edit►Select All**. Toggles the *select all* state to on; **Cut**, **Copy**, and delete operations then apply to the entire model.

virtual void handleSelectAll(Boolean);	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---	-----------------	----------------	-------------------

Constructors

The framework handles construction and destruction for objects of this class.

IGUIBundle This constructor is called by the framework factory function `IComponentStationery::createBundle` (p. 15).

If you derive a class from `IGUIBundle`, your derived class's constructor will explicitly or implicitly call this default constructor.

IGUIBundle();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------	-----------------	----------------	-------------------

~IGUIBundle

~IGUIBundle();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
----------------	-----------------	----------------	-------------------

Informational

Use these members to obtain references to the different parts of a component.

component Returns a reference to the GUI bundle's associated component.

IComponent& component() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-----------------------------------	-----------------	----------------	-------------------

frameWindow

Returns a reference to the out-of-place frame window. The frame window is hidden whenever a component is in-place activated.

IFrameWindow& frameWindow() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---------------------------------------	-----------------	----------------	-------------------

objectMenuBar

Returns a reference to the in-place menu bar if it is currently active, or else the out-of-place menu bar.

IMenuBar& objectMenuBar();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-------------------------------	-----------------	----------------	-------------------

objectToolBar

Returns a reference to the in-place toolbar if it is currently active, or else the out-of-place toolbar.

IToolBar& objectToolBar();	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-------------------------------	-----------------	----------------	-------------------

objectView Returns a reference to the in-place view if it is currently active, or else the out-of-place view.

IView& objectView() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-------------------------------	-----------------	----------------	-------------------

view Returns a reference to the out-of-place view even if an in-place view is currently active.

IView& view() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-------------------------	-----------------	----------------	-------------------

IGUIBundle

Protected Functions

Helper Methods

Use this member to obtain information about the status of the file.

okToDiscardFile

Returns whether it is okay to discard the file. Returns true if the file is unchanged, or if the end-user selected No in response to a **Do you want to save...** dialog.

```
virtual Boolean  
    okToDiscardFile();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



IMGrabbable

Derivation Inherits from none.

Inherited By IEmbeddedComponent

Header File igrabhdl.hpp

Members	Member	Page	Member	Page
	doDragDrop	437	rect	438
	inWindow	438	setRect	438

IMGrabbable is an abstract mixin class that allows objects to display move/resize handles. This class works with class IGrabHandles (p. 339) to implement the grabbed object's side of the move/resize protocol.

The Compound Document Framework's embedded components inherit their move/resize functionality from this class

Public Functions

Control

Use this member to handle drag and drop events.

doDragDrop Called by the grab handles when the user clicks the left mouse button. This function should initiate a drag operation and take over handling of mouse events. This function is only called if the grab handles are constructed with dragMove equal to true. See IGrabHandles::IGrabHandles (p. 339) for more information.

virtual Boolean	Win	PM	Motif
doDragDrop(const IPoint& startPoint);	<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IPoint& The start point of the mouse before the drag operation.

Location/Positioning

Use these members to obtain location information for, and set positions of, grabbable objects.

IMGrabbable

inWindow Returns the window in which the grabbable object, and therefore the grab handles, will appear.

IMGrabbable derived classes must implement this function.

```
virtual IWindow&
    inWindow() const = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

rect Returns the grabbable object's area in relative coordinates of the window returned by IMGrabbable::inWindow. Depending on the type of grabbable object, this is either the object's physical display area or the bounding rectangle of the object. For example, a grabbable ellipse would provide its bounding rectangle.

IMGrabbable derived classes must implement this function.

```
virtual IRectangle
    rect() const = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

setRect Moves or resizes the grabbable object. This function is called by the grab handles whenever the grabbable object is moved or resized. Your derived class of IMGrabbable is responsible for moving the object to the new window position and repainting the window. The derived class's override of this function generally needs to call IGrabHandles::draw (p. 341) to redraw the handles on top of the object once it has been moved.

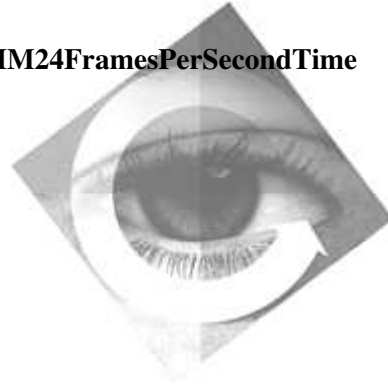
IMGrabbable derived classes must implement this function.

```
virtual void
    setRect( const IRectangle& newRect ) = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IRectangle&
The new location.



IMM24FramesPerSecondTime

Derivation


- IBase
- IVBase
- IMMTime
- IMMHourMinSecFrameTime
- IMM24FramesPerSecondTime

Inherited By None.

Header File immtime.hpp

Members	Member	Page
	Constructor	439
	operator unsigned long	440
	~IMM24FramesPerSecondTime	440

The IMM24FramesPerSecondTime data type class represents the frame-numbering system that assigns a number to each frame of video. This system was developed by the Society of Motion Picture and Television Engineers. The 8-digit code is in the form HH:MM:SS.FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point. This time format is based on 24 frames per second.


 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMM24FramesPerSecondTime

 IMM24FramesPerSecondTime(unsigned long value = defaultTime); Win PM Motif
Y Y N

You can construct an IMM24FramesPerSecondTime from the following parameter variable:

IMM24FramesPerSecondTime

value A time value where:

- 1. 1st byte is the frames
- 2. 2nd byte is the seconds
- 3. 3rd byte is the minutes
- 4. 4th byte is the hours

2

IMM24FramesPerSecondTime(
const IMM24FramesPerSecondTime& time);

Win

PM

Motif

Y

Y

N

~IMM24FramesPerSecondTime

virtual

~IMM24FramesPerSecondTime();

Win

PM

Motif

Y

Y

N

Conversions

Use these members to cast the time to an unsigned long.

operator
unsigned
long

Returns the time as a unsigned long in the following format (FFSSMMHH):

- 1. 1st byte is the frames
- 2. 2nd byte is the seconds
- 3. 3rd byte is the minutes
- 4. 4th byte is the hours

virtual

operator unsigned long() const;

Win

PM

Motif

Y

Y

N

Inherited Public Functions

IMMHourMinSecFrameTime		
asString	hours	ordinal
frames	minutes	seconds
framesPerSecond	operator unsigned long	setTimeToOrdinal

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=

IMM24FramesPerSecondTime

IMMTime		
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMM25FramesPerSecondTime

IMM25FramesPerSecondTime

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IMMTIME
    IMMTIME --> IMMHourMinSecFrameTime
    IMMHourMinSecFrameTime --> IMM25FramesPerSecondTime
```

Inherited By None.

Header File immtime.hpp

Members	Member	Page
	Constructor	442
	operator unsigned long	443
	~IMM25FramesPerSecondTime	443

The IMM25FramesPerSecondTime data type class represents the frame-numbering system that assigns a number to each frame of video. This system was developed by the Society of Motion Picture and Television Engineers. The 8-digit code is in the form HH:MM:SS.FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point. This time format is based on 25 frames per second.




The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMM25FramesPerSecondTime

 IMM25FramesPerSecondTime(unsigned long value = defaultTime); Win PM Motif
Y Y N

You can construct an IMM25FramesPerSecondTime from the following parameter variable:

IMM25FramesPerSecondTime

value A time value where:

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

2	IMM25FramesPerSecondTime(const IMM25FramesPerSecondTime& time);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

~IMM25FramesPerSecondTime

<i>virtual</i> ~IMM25FramesPerSecondTime();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long Returns the time as a unsigned long in the following format (FFSSMMHH):

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

<i>virtual</i> operator unsigned long() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Inherited Public Functions

IMMHourMinSecFrameTime		
asString	hours	ordinal
frames	minutes	seconds
framesPerSecond	operator unsigned long	setTimeToOrdinal

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=

IMM25FramesPerSecondTime

IMMTime		
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

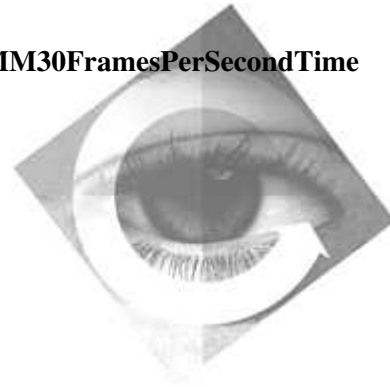
IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMM30FramesPerSecondTime

Derivation

```


IBase
IVBase
IMMTime
IMMHourMinSecFrameTime
IMM30FramesPerSecondTime
  
```

Inherited By None.

Header File immtime.hpp

Members	Member	Page
	Constructor	445
	operator unsigned long	446
	~IMM30FramesPerSecondTime	446

The IMM30FramesPerSecondTime data type class represents the frame-numbering system that assigns a number to each frame of video. This system was developed by the Society of Motion Picture and Television Engineers. The 8-digit code is in the form HH:MM:SS.FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point. This time format is based on 30 frames per second.


 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMM30FramesPerSecondTime

 IMM30FramesPerSecondTime(unsigned long value = defaultTime); Win PM Motif
Y Y N

You can construct an IMM30FramesPerSecondTime from the following parameter variable:

IMM30FramesPerSecondTime

value A time value where:

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

```
2 IMM30FramesPerSecondTime(  
    const IMM30FramesPerSecondTime& time);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IMM30FramesPerSecondTime

```
virtual  
~IMM30FramesPerSecondTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long Returns the time as a unsigned long in the following format (FFSSMMHH):

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

```
virtual  
operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMHourMinSecFrameTime		
asString	hours	ordinal
frames	minutes	seconds
framesPerSecond	operator unsigned long	setTimeToOrdinal

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=

IMM30FramesPerSecondTime

IMMTime		
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File immamix.hpp

Members	Member	Page	Member	Page
	Constructor	448	openOnThread	450
	balance	450	pitch	451
	bass	450	setBalance	451
	close	449	setBass	452
	disableMonitoring	454	setGain	452
	enableMonitoring	454	setPitch	452
	gain	451	setTreble	453
	isMonitoringEnabled	455	treble	453
	isOpenStringValid	456	~IMMampMixer	449
	open	449		

The IMMampMixer class provides input or output switching and sound-shaping services, such as volume, treble, gain, or bass control. The amplifier-mixer is similar to a home stereo amplifier and mixer. Devices are connected to the IMMampMixer so that audio signals can be transferred to speakers or another device.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMampMixer



```
IMMampMixer( Boolean openNow = true,
              unsigned long instance = 0,
              Boolean openShareable = true );
```

Win	PM	Motif
Y	Y	N

IMMAmpMixer

You can construct an IMMAmpMixer from the following:

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.

instance Specifies your own instance number instead of one being generated for you.

openShareable
If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

```
2 IMMAmpMixer( unsigned long deviceIdentifier,           Win PM Motif
               const IString& newAlias = IString ( ) );   Y   Y   N
```

A derived class can construct an IMMAmpMixer from the following:

deviceIdentifier
The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

~IMMAmpMixer

```
virtual                                           Win PM Motif
~IMMAmpMixer();                                Y   Y   N
```

Implementation

These members check if the passed in string is in the correct format to open the current device.

close Closes a device.

```
virtual IMMAmpMixer&                               Win PM Motif
close( CallType call = wait );                       Y   N   N
```



This member is overridden in this derived class for specific operating system behavior.

open Opens the device.

```
virtual IMMAmpMixer&                               Win PM Motif
open( const IString& fileOrDevice = IString ( ),    Y   N   N
      Boolean shareable = true,
      CallType call = wait );
```

IMMAmpMixer



This member is overridden in this derived class for specific operating system behavior.

openOnThread

Opens the device.

```
virtual IMMAmpMixer&
    openOnThread( const IString& fileOrDevice = IString ( ),           Win PM Motif
                  Boolean shareable = true,                          Y   N   N
                  CallType call = wait );
```



This member is overridden in this derived class for specific operating system behavior.

Mixer Attributes

Use these members to query and set the different amplifier-mixer attributes.

balance

Returns the balance, where 0 is defined as full left balance and 100 is defined as full right balance.

```
unsigned long
    balance( CallType call = wait ) const;           Win PM Motif
                                                    Y   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

bass

Returns the bass, where 0% is the least amount of bass and 100% is the most amount of bass.

```
unsigned long
    bass( CallType call = wait ) const;             Win PM Motif
                                                    Y   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.

IMMAmpMixer

Exceptions	
InvalidRequest	The device must be in the open state before calling this function.

gain

Returns the gain, where 0% is the least amount of gain and 100% is the most amount of gain..

```
unsigned long                                     Win PM Motif  
gain( CallType call = wait ) const;              Y   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

pitch

Returns the pitch, where 0 is the lowest pitch and 100 the highest pitch.

```
unsigned long                                     Win PM Motif  
pitch( CallType call = wait ) const;              Y   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setBalance

Sets the balance, where 0 is defined as full left balance and 100 is defined as full right balance.

```
virtual IMMAmpMixer&                             Win PM Motif  
setBalance( unsigned long balance,                Y   Y   N  
            CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMAmpMixer

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setBass Sets the bass, where 0% is the least amount of bass and 100% is the most amount of bass.

```
virtual IMMAmpMixer&  
    setBass( unsigned long bass,  
            CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setGain Sets the gain, where 0% is the least amount of gain and 100% is the most amount of gain.

```
virtual IMMAmpMixer&  
    setGain( unsigned long gain,  
            CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setPitch Sets the pitch, where 0 is lowest pitch and 100 is the highest pitch.

IMMAmpMixer

```
virtual IMMAmpMixer&  
    setPitch( unsigned long pitch,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setTreble

Sets the treble, where 0 is the least amount of treble and 100 is the most amount of treble.

```
virtual IMMAmpMixer&  
    setTreble( unsigned long treble,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

treble

Returns the treble, where 0 is the least amount of treble and 100 is the most amount of treble.

```
unsigned long  
    treble( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

IMMAmpMixer

Monitoring

Use these members to either allow or not allow the signal from an input device to be heard when it is being routed to another device.

The amplifier-mixer device has the ability to monitor (listen to) the audio signal from one device while the audio signal from a second device is being recorded. The main use of monitoring is for recording and mixing more than one audio source. The amplifier-mixer device supports one audio connection and the ability to monitor a second audio source.

disableMonitoring

Does not allow the signal from an input device to be heard as it is being routed to another device. The amplifier-mixer device has the ability to monitor (listen to) the audio signal from one device while the audio signal from a second device is being recorded. The main use of monitoring is for recording and mixing more than one audio source. The amplifier-mixer device supports one audio connection and the ability to monitor a second audio source.

```
virtual IMMAmpMixer&
    disableMonitoring( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

enableMonitoring

Allows the signal from an input device to be heard as it is being routed to another device. The amplifier-mixer device has the ability to monitor (listen to) the audio signal from one device while the audio signal from a second device is being recorded. The main use of monitoring is for recording and mixing more than one audio source. The amplifier-mixer device supports one audio connection and the ability to monitor a second audio source.

```
virtual IMMAmpMixer&
    enableMonitoring( Boolean enable = true,
                      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMAmpMixer

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

isMonitoringEnabled

Returns true if the monitor is turned on. The amplifier-mixer device has the ability to monitor (listen to) the audio signal from one device while the audio signal from a second device is being recorded.

Boolean	Win	PM	Motif
isMonitoringEnabled(CallType call = wait) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

Inherited Public Functions

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming

IMMAmpMixer

IMMDevice		
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Implementation

These members check if the passed in string is in the correct format to open the current device.

isOpenStringValid

Returns true if the passed in open string is valid for this device.

```
virtual Boolean
  isOpenStringValid( const IString& deviceName ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

open

Opens the device.

```
virtual IMMAmpMixer&
  open( unsigned long instanceNumber,
        Boolean shareable = true,
        CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	N	N

IMMAmpMixer



This member is overridden in this derived class for specific operating system behavior.

Opens the device.

openOnThread

```
virtual IMMAmpMixer&
  openOnThread( unsigned long instanceNumber,
                Boolean shareable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



This member is overridden in this derived class for specific operating system behavior.

Inherited Protected Functions

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer

IMMAmpMixer

IMMDevice		
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMAudioBuffer

Derivation IBase
IMMAudioBuffer

Inherited By None.

Header File immabuf.hpp

Members	Member	Page	Member	Page
	Constructor	462	operator =	462
	audio	464	samplesPerSecond	460
	bitsPerSample	460	setBitsPerSample	460
	blockAlignment	460	setBlockAlignment	461
	bytesPerSecond	460	setBytesPerSecond	461
	channels	460	setChannels	461
	contentType	463	setContentType	463
	data	461	setData	462
	format	463	setFormat	463
	headerData	461	setMediaType	464
	length	462	setSamplesPerSecond	461
	mediaType	464	~IMMAudioBuffer	463

The IMMAudioBuffer class provides a buffering mechanism that allows device classes to operate on a buffer as if it were a file. It is recommended that you set up the type and format of audio data that is to be recorded or playing. Using buffers improves the performance of multimedia applications that perform numerous file input and output operations when accessing media devices. You might want to use a buffer for applications that perform many input and output operations that are less than 4KB each. However, if your operations are larger than 4KB, then you might not want to use a buffer. In this situation, you will see little to no performance gain. You might have performance degradation, because your application will have to copy data in big blocks at a time.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Audio Attributes

Use these members to query and set the audio attributes for the audio buffer.

IMMAudioBuffer

bitsPerSample

Returns the number of bits-per-sample. This is the number of bits of data used to represent each sample of each channel. The standard values are 4, 8, and 16.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bitsPerSample() const;	<i>N</i>	<i>Y</i>	<i>N</i>

blockAlignment

Returns the block alignment of data in bytes. The system processes a multiple of block-aligned bytes of data at a time. For example, if you have a 24-byte buffer, then you can only set a block alignment of 2, 3, 4, 6, 8, or 12.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
blockAlignment() const;	<i>N</i>	<i>Y</i>	<i>N</i>

bytesPerSecond

Returns the average number of bytes-per-second played or recorded. This is the rate the data is transferred to the hardware.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bytesPerSecond() const;	<i>N</i>	<i>Y</i>	<i>N</i>

channels

Returns the number of audio channels set. For example, mono is 1; stereo is 2.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
channels() const;	<i>N</i>	<i>Y</i>	<i>N</i>

samplesPerSecond

Returns the number of samples-per-second played or recorded. This is the sampling rate, in kilohertz, which each channel should use. Standard values are 11025, 22050, and 44100.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
samplesPerSecond() const;	<i>N</i>	<i>Y</i>	<i>N</i>

setBitsPerSample

Sets the bits-per-sample to be played or recorded. This is the number of bits of data used to represent each sample of each channel. The standard values are 4, 8, and 16.

IMMAudioBuffer

IMMAudioBuffer& setBitsPerSample(unsigned long bitsPerSample);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

setBlockAlignment

Sets the block alignment of the data. The system processes a multiple of block-aligned bytes of data at a time. For example, if you have a 24-byte buffer, then you can only set a block alignment of 2, 3, 4, 6, 8, or 12.

IMMAudioBuffer& setBlockAlignment(unsigned long alignment);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

setBytesPerSecond

Sets the average bytes per second to be played or recorded. This is the rate the data is transferred to the hardware.

IMMAudioBuffer& setBytesPerSecond(unsigned long averageBytes);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

setChannels Sets the number of audio channels for playing and recording. Monaural is 1; stereo is 2.

IMMAudioBuffer& setChannels(unsigned long channels);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

setSamplesPerSecond

Sets the sampling rate for playing and recording. This is the sampling rate, in kilohertz, which each channel should use. Standard values are 11025, 22050, and 44100.

IMMAudioBuffer& setSamplesPerSecond(unsigned long samplesPerSecond);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

Buffer Management

Use these members to query and set the buffer data and to query the length of the buffer.

data Returns an IString that contains a copy of the data in the audio buffer.

IString data() const;	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--------------------------	------------------------	-----------------------	--------------------------

headerData Returns an IString that contains the operating system specific header information.

IMMAudioBuffer

	IStrng headerData() const;	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
length	Returns the size of the buffer.			
	unsigned long length() const;	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
setData	Sets the contents and size of the audio buffer to those of the passed-in buffer.			
	IMMAudioBuffer& setData(const void* data, unsigned long dataLength);	<u>Win</u> <i>N</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>

Constructors

You can construct, copy, and destruct objects of this class.

IMMAudioBuffer

1	IMMAudioBuffer(const IMMAudioBuffer& audioBuffer);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	------------------------	-----------------------	--------------------------

2	IMMAudioBuffer(void* audioData, unsigned long dataLength);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	------------------------	-----------------------	--------------------------

You can construct an IMMAudioBuffer from the following:

audioData Initialize the audio buffer with this.

dataLength
Initialize the size of the audio buffer to this.

3	IMMAudioBuffer(unsigned long dataLength = 0);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	------------------------	-----------------------	--------------------------

You can construct an IMMAudioBuffer from the following:

dataLength
The size of the audio buffer to be created. It is initialized with all 0's.

operator = Copies the audio buffer information from the passed in audio buffer.

IMMAudioBuffer

IMMAudioBuffer& operator =(const IMMAudioBuffer& audioBuffer);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

~IMMAudioBuffer

~IMMAudioBuffer();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--------------------	-----------------	----------------	-------------------

Content Type

Use these members to set and query the quality of the audio content.

contentType Returns the audio content type.

ContentType contentType() const;	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
-------------------------------------	-----------------	----------------	-------------------

setContentType

Sets the audio content type.

IMMAudioBuffer& setContentType(ContentType contentType);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Format

Use these members to set and query the format of the audio data.

format Returns the interpretation of the audio format. This is what format the audio data is in.

Format format() const;	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
---------------------------	-----------------	----------------	-------------------

setFormat Sets the audio format to be used. This determines what format the audio data is in.

IMMAudioBuffer& setFormat(Format tag = pcm);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Media Type

Use these members to query and set the type of media in the buffer.

IMMAudioBuffer

mediaType Returns the type of multimedia data in the buffer.

```
unsigned long  
mediaType() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

setMediaType Sets the type of multimedia data in the buffer.

```
IMMAudioBuffer&  
setMediaType( unsigned long media = audio );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Public Data

Media Type

Use these members to query and set the type of media in the buffer.

audio Represents audio data.

```
static const unsigned long  
audio;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ContentType ContentType {
 Unknown,
 Voice,
 Music,
 HiFi = 0x0004
};

IMMAudioBuffer

Use this enumeration for determining the quality of the audio content. Valid values are the following:

unknown

Represents unknown audio quality.

voice

Represents voice (limited range) quality.

music

Represents FM radio or equivalent quality.

hifi

Represents high quality.

Format

```
Format {  
    pcm = 0x0001,      adpcm,      ibmcvsd = 0x0005,  alaw,  
    mulaw,             okiadpcm = 0x0010, dviadpcm,    digistd = 0x0015,  
    digifix,           ibmmulaw = 0x0101, ibmalaw,     avcadpcm,  
    ctadpcm = 0x0200  
};
```

Use this enumeration for determining how audio information is stored and interpreted. The pcm format is the most commonly used format. Valid values are the following:

pcm

Represents pulse code modulation. This refers to a variation of a digital signal to represent audio amplitude. This method of assigning binary values to amplitude levels supports the conversion of analog signals to digital signals by sound cards.

adpcm

Represents adaptive differential pulse code modulation. This is a technique for compressing waveform samples. It can reduce the amount of data storage required by a factor of 16 to 1, but some price is paid in fidelity for the higher compression rates.

ibmcvsd

Represents the IBM Speech Viewer format.

alaw

Represents the CCITT A-Law format.

mulaw

Represents the CCITT MuLaw format.

okiadpcm

Represents the OKI adaptive pulse code modulation format.

IMMAudioBuffer

dviadpcm

Represents the DVI adaptive pulse code modulation format.

digistd

Represents the IBM Digispeech standard format.

digifix

Represents the IBM Digispeech fixed format.

ibmmulaw

Represents the IBM MuLaw format.

ibmalaw

Represents the IBM A-Law format.

avcadpcm

Represents IBM AVC adaptive differential pulse code modulation format.

ctadpcm

Represents the Creative Labs adaptive differential pulse code modulation format.



IMMAudioCD

Derivation

- IBase
- IVBase
- INotifier
- IStandardNotifier
- IMMDevice
- IMMPlayableDevice
- IMMRemovableMedia
- IMMAudioCD

Inherited By None.

Header File immcdda.hpp

Members	Member	Page	Member	Page
	Constructor	471	profile	477
	contents	477	resume	474
	disableAutoPlay	468	setDiscTitle	470
	disableContinuousPlay	468	setProfile	477
	discId	468	setProgram	478
	discProgramKey	482	setTrackTitle	470
	discTitle	468	setVolume	472
	discTitleKey	482	startScanningBackward	475
	enableAutoPlay	469	startScanningForward	475
	enableContinuousPlay	469	stop	476
	getTableFromDisc	481	trackBackward	476
	goToEntry	472	trackForward	477
	isAutoPlayEnabled	469	trackStartedId	482
	isContinuousPlayEnabled	469	trackTitle	470
	isOpenStringValid	480	uniqueDiscIdentifier	480
	numberOfTracks	469	upc	471
	pause	473	volume	472
	play	474	~IMMAudioCD	472
	positionTimerId	482		

The IMMAudioCD class is the audio compact disc (CD) class and provides behavior for CD audio players. Most people are familiar with a home or car CD player. So, just as for a normal CD player, there are functions for play, pause, stop, trackForward, trackBackward, scanning, and so on. Plus, there are some additional features, for example, rearranging the playback order for tracks and storing track and disc titles.



The multimedia interface classes are not supported on the Win32s platform.

IMMAudioCD

Public Functions

Compact Disc Information

Use these members to query information and states for the CD audio device and the loaded compact disc. For example, you can query and set `autoPlay` and `continuousPlay`, and you can query the number of tracks, the disc title, the disc identifier, and the universal product code (`upc`).

disableAutoPlay

Sets autoplay off. Autoplay causes the CD audio device to start playing whenever a CD is loaded.

IMMAudioCD& disableAutoPlay();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
-----------------------------------	------------------------	-----------------------	--------------------------

disableContinuousPlay

Sets continuous play off. If the continuous play is set, then the current track is repeated continuously on playback.

IMMAudioCD& disableContinuousPlay();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

discId

Returns the identifier of the current disc. If there is no identifier on the CD, then it returns a null `IString`.

<code>IString</code> discId(CallType call = wait) const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

Exceptions	
<code>IAccessError</code>	The device identifier is not valid; possibly the device is closed.
<code>IAccessError</code>	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
<code>IAccessError</code>	An internal driver error occurred.
<code>IInvalidRequest</code>	The device must be in the open state before calling this function.
<code>IInvalidRequest</code>	There must be media present in the device.

discTitle

Returns the disc title. A null `IString` is returned if you have not set a title.

IMMAudioCD

NSString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
discTitle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	There must be media present in the device.
InvalidRequest	The device must be in the open state before calling this function.

enableAutoPlay

Turns autoplay on or off. Autoplay causes the audio CD device to start playing whenever a CD is loaded.

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
enableAutoPlay(Boolean autoPlay = true);	<i>Y</i>	<i>Y</i>	<i>N</i>

enableContinuousPlay

Turns continuous play on or off. If the continuous play is set, then the current track is repeated continuously on playback.

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
enableContinuousPlay(Boolean continuousPlay = true);	<i>Y</i>	<i>Y</i>	<i>N</i>

isAutoPlayEnabled

Returns true, if autoplay is turned on. Autoplay causes the audio CD device to start playing whenever a CD is loaded.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isAutoPlayEnabled() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isContinuousPlayEnabled

Returns true, if continuous play is turned on. If the continuous play is set, then the current track is repeated continuously on playback.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isContinuousPlayEnabled() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

numberOfTracks

Returns the number of tracks.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
numberOfTracks(CallType call = wait) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

IMMAudioCD

setDiscTitle Sets the disc's title. The disc title is stored in the currently set profile. See `setProfile` (p. 477) for more information on profiles.

```
IMMAudioCD&
    setDiscTitle( const IString& title );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IInvalidRequest	There must be media present in the device.
IInvalidRequest	The device must be in the open state before calling this function.

setTrackTitle Sets the track's title. The track title is stored in the currently set profile. See `setProfile` (p. 477) for more information on profiles.

```
IMMAudioCD&
    setTrackTitle( const IString& title,
                  unsigned long track );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

trackTitle Returns the track's title. A null `IString` is returned if you have not set a title for the passed-in track.

```
IString
    trackTitle( unsigned long track ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

IMMAudioCD

upc

Returns the disc's universal product code (serial number). If the disc does not contain a upc, then a null IString is returned.

```
IString  
upc( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There must be media present in the device.

Constructors

You can construct and destruct objects of this class.

IMMAudioCD

```
IMMAudioCD( IProfile* profile = 0,  
            Boolean openNow = true,  
            unsigned long instance = 0,  
            Boolean openShareable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMAudioCD from the following:

- profile* You can provide your own profile where information can be retrieved and stored from. Some of the information stored in a profile is disc titles and the IMMAudioCDContents for different CDs. If you pass in 0, then the device uses the system profile for storing and retrieving information.
- openNow* If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.
- instance* You can provide your own instance number instead of letting IMMAudioCD generate one.
- openShareable* If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

IMMAudioCD

~IMMAudioCD

<pre>virtual ~IMMAudioCD();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Implementation

These members are overridden to provide additional processing.

setVolume Sets the volume of the audio channel for the device to a percent (from 0% to 100%) of the maximum audio level.

<pre>virtual IMMAudioCD& setVolume(unsigned long volume, AudioChannel channel = all, const IMMMillisecondTime& over = IMMMillisecondTime (), CallType call = wait);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					



This member is overridden in this derived class for specific operating system behavior.

volume Returns the volume of the audio channel for the device as a percent (from 0% to 100%) of the maximum audio level.

<pre>virtual unsigned long volume(AudioChannel channel = left, CallType call = wait) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					



This member is overridden in this derived class for specific operating system behavior.

Play, Stop, and Scan

Use these members to stop playing or scanning, start scanning or playing, resume or pause the playback, or move the current position to a new location.

goToEntry Moves the current position to the passed in location. This location is a track number and the time into the track.

	<pre>virtual IMMAudioCD& goToEntry(unsigned long index);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Moves to the track stored in the current program at the passed-in index. If you passed in 3, it looks up the track number in the current program at that location. If you have not set your own program then the default program is used. This program is a straight mapping of the tracks; for example, at index 3, track number 3 is stored.

IMMAudioCD

You can set your own program. For example, you could have the following: at index 1, track 5; at index 2, track 4; and at index 3, track 6. If you passed in index 3, this function moves the current position to track number 6.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

2	virtual IMMAudioCD& goToEntry(IMMAudioCDContents::Cursor cursor);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	-------------------------------	------------------------------	---------------------------------

Moves to the track stored in the current program at the passed-in cursor. If you passed in a cursor that points to index 3, it looks up the track number in the current program at that location. If you have not set your own program then the default program is used. This program is a straight mapping of the tracks; for example, at index 3, track number 3 is stored. You can set your own program. For example, you could have the following: at index 1, track 5; at index 2, track 4; and at index 3, track 6. If you passed in a cursor that points to index 3, this function moves the current position to track number 6.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There must be media present in the device.

pause

Pauses the audio CD device if the audio CD device is playing.

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
virtual IMMAudioCD& pause(CallType call = wait);	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMAudioCD

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

play

Starts playing the CD device from the passed-in start position to the passed-in end position. If *from* is omitted, the CD device starts playing at the current position; if *to* is omitted, play stops at the end of the disc. If the *resumeIfPaused* Boolean is true and the CD player is paused, it resumes playing at the current location; otherwise, it starts playing at the passed-in *from* location.

```
virtual IMMAudioCD&
    play( const IMMTime& from = IMMTime ( ),
          const IMMTime& to = IMMTime ( ),
          Boolean resumeIfPaused = true,
          CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the contents of the CD.

resume

Resumes playback of the compact disc from a paused state. The previous specified *to* parameter in the play (p. 474) function remains in effect.

```
virtual IMMAudioCD&
    resume( Boolean resume = true,
            CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IMMAudioCD

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There must be media present in the device.

startScanningBackward

Causes the audio CD device to search backward at high speed. If the device is currently scanning backward, this function has no effect, but, if the device is currently scanning forward, this causes the scanning to stop and to start scanning backward. You can stop the scanning by calling the stop (p. 476) function, or, if the CD was playing when scanning started, the play function stops the scanning and resumes playback at the new location. The scanning wraps to the end when it reaches the beginning of the compact disc.

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
startScanningBackward();	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There must be media present in the device.

startScanningForward

Causes the audio CD device to search forward at high speed. If the device is currently scanning forward, this function has no effect, but, if the device is currently scanning backward, this causes the scanning to stop and to start scanning forward. You can stop the scanning by calling the stop (p. 476) function, or, if the CD was playing when scanning started, the play function stops the scanning and resumes playback at the new location. The scanning wraps to the beginning when it reaches the end of the compact disc.

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
startScanningForward();	Y	Y	N

IMMAudioCD

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

stop

Stops playback and backward or forward scanning of the compact disc.

```
virtual IMMAudioCD&  
    stop( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

trackBackward

Moves the current position backwards the passed-in number of tracks. If the device was playing, it starts playing at the new position.

```
IMMAudioCD&  
    trackBackward( unsigned long decrement = 1 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

IMMAudioCD

trackForward Moves the current position forwards the passed-in number of tracks. If the device was playing, it starts playing at the new position.

```
IMMAudioCD&
    trackForward( unsigned long increment = 1 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There must be media present in the device.

Profile and Program

Use these members to change and query the profile and program.

contents Returns a copy of the CD's table of contents.

```
IMMAudioCDContents
    contents() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

profile Returns the current profile. The profile contains one or more compact disc titles and their IMMAudioCDContents. This can be useful if you want to retain a table of contents for one or more compact discs. For example, you might have a demo CD that you want to play the third track first, followed by the second, and so on. You would not want to have to rearrange the tracks every time you loaded that CD. By storing the information in a profile, the IMMAudioCD class can load that disc's information from the profile so you do not have to reenter anything.

```
IProfile
    profile() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setProfile Sets a new profile. The profile contains one or more compact disc titles and their IMMAudioCDContents. This can be useful if you want to retain a table of contents for one or more compact discs. For example, you might have a demo CD that you want to play the third track first, followed by the second, and so on. You would not want to have to rearrange the tracks every time you loaded that CD. By storing the information in a profile, the IMMAudioCD class can load that disc's information from the profile so you do not have to reenter anything.

IMMAudioCD

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setProfile(IProfile& profile);	<i>Y</i>	<i>Y</i>	<i>N</i>

setProgram Sets a new table of contents. The table of contents is stored in the currently set profile. The IMMAudioCD class plays and scans the compact disc according to the current table of contents. This allows you create your own table of contents for the current disc. For instance, you might want to store your compact discs (title and table of contents) in a database. Then, when you insert a CD, you could query its ID and see if it is in your database. If so, then you could retrieve your table of contents and set it into the device so that it always plays back the same way.

IMMAudioCD&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setProgram(const IMMAudioCDContents& program);	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

Inherited Public Functions

IMMRemovableMedia		
closeDoor	isMediaPresent	openDoor
enableNotification	lockDoor	unlockDoor

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume

IMMAudioCD

IMMDevice		
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class.

IMMAudioCD

```
IMMAudioCD( unsigned long deviceIdentifier,
             const IString& newAlias = IString ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMAudioCD from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Implementation

These members are overridden to provide additional processing.

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

```
virtual Boolean
  isOpenStringValid( const IString& deviceName ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

uniqueDiscIdentifier

Returns the unique disc identifier. This value is the name of this class plus the identifier of the disc. This can be useful to use in addition to the upc, because not all manufacturers put a upc on their CDs.

```
virtual IString
  uniqueDiscIdentifier() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

Table

Use these members to query the hard-coded table of contents stored on the compact disc.

IMMAudioCD

getTableFromDisc

Generates a table of contents based on the hard-coded values stored on the compact disc. The only data stored on the compact disc are the tracks and their lengths.

IMMAudioCDContents	<u>Win</u>	<u>PM</u>	<u>Motif</u>
getTableFromDisc();	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There must be media present in the device.

Inherited Protected Functions

IMMRemovableMedia		
notificationHandler	setNotificationHandler	

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

IMMAudioCD

Public Data

Compact Disc Information

Use these members to query information and states for the CD audio device and the loaded compact disc. For example, you can query and set autoPlay and continuousPlay, and you can query the number of tracks, the disc title, the disc identifier, and the universal product code (upc).

discProgramKey

Contains the text string "IMMAudioCDProgramTitle". This string is used as a key to query and retrieve information from a profile.

static const IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
discProgramKey;	Y	Y	N

discTitleKey

This is set to the text string "IMMAudioCDDiscTitle". This string is used as a key to query and retrieve information from a profile.

static const IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
discTitleKey;	Y	Y	N

Notification Event Descriptions

These INotificationId strings are used for all notifications that IMMAudioCD provides to its observers.

positionTimerId

Notification identifier provided to observers as the position changes. IMMAudioCD provides a pointer to an IMMTrackMinSecFrameTime (p. 689) in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value represents the current position of the CD.

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
positionTimerId;	Y	Y	N

trackStartedId

Notification identifier provided to observers when the a new track is started. IMMAudioCD provides a pointer to an IMMTrackMinSecFrameTime (p. 689) in the eventData field of the INotificationEvent (Vol. I). This value represents the current position of the CD.

IMMAudioCD

```
static INotificationId const  
    trackStartedId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

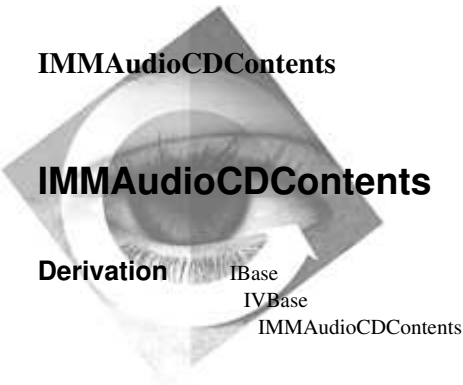
IMMRemovableMedia		
mediaLoadedId		

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	




Inherited By None.

Header File immcdda.hpp

Members				
Member	Page	Member	Page	
Constructor	484	numberOfTracks	487	
addEntryAsFirst	485	operator =	485	
addEntryAsNext	485	owner	487	
controlInformation	486	removeEntryAt	485	
country	486	replaceEntryAt	486	
discId	486	serialNumber	487	
endOfTrack	486	startOfTrack	487	
isValid	487	track	487	
numberOfEntries	485	~IMMAudioCDContents	485	

The IMMAudioCDContents class is the class for an audio CD's table of contents. It allows you to keep a list of tracks. With this list you can rearrange the playback order of the tracks. For example, you could put track 5 at the first location, track 7 next, and then track 1.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct, copy, and destruct objects of this class.

IMMAudioCDContents

Use this function to create an IMMAudioCDContents.

IMMAudioCDContents(const IMMAudioCDContents& newToc);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

You cannot create your own table of contents without asking for one from the IMMAudioCD class.

IMMAudioCDContents

operator = Sets the contents to be the same as another table of contents.

IMMAudioCDContents& operator =(const IMMAudioCDContents& newContents);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

~IMMAudioCDContents

virtual ~IMMAudioCDContents();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
-----------------------------------	-----------------	----------------	-------------------

Entry Information

Use these members to add, remove, and replace the track numbers in the table of contents. For example, you want to listen to all of the CD except track 2 and you want to listen to track 5 played twice. To do this, you create a cursor on the table of contents. Then, you move the cursor to track 2 and call the remove entry function. Next, you move the cursor to track 5 and call the add entry function with 5 for the track number.

addEntryAsFirst

Adds the track number to the beginning of the playback list.

IMMAudioCDContents& addEntryAsFirst(unsigned long trackNumber);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

addEntryAsNext

Adds the track number after the cursor in the playback list.

IMMAudioCDContents& addEntryAsNext(unsigned long trackNumber, const Cursor& cursor);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

numberOfEntries

Returns the number of tracks in the playback list.

unsigned long numberOfEntries() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

removeEntryAt

Removes the track number at the cursor from the playback list.

IMMAudioCDContents

IMMAudioCDContents& removeEntryAt(const Cursor& cursor);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

replaceEntryAt

Replaces the track number at the cursor.

IMMAudioCDContents& replaceEntryAt(unsigned long newTrackNumber, const Cursor& cursor);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Exceptions	
IInvalidParameter	The new track number can not be less than 1 or greater than the number of tracks.
IInvalidParameter	The passed in cursor or this object is not valid.
IInvalidRequest	An internal processing error occurred with the stored list of tracks.

Track Information

Use these members to query information about the entry at the given cursor for this table of contents.

controlInformation

Returns the track control information at the given cursor. This value cannot be set because it is set by the manufacturer of the CD.

unsigned long controlInformation(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

country

Returns the country at the given cursor. This value cannot be set because it is set by the manufacturer of the CD.

unsigned long country(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

discId

Returns the compact disc identifier.

IString discId() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------------------------	-----------------	----------------	-------------------

endOfTrack

Returns the ending time (position) of the track.

IMMTime endOfTrack(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IMMAudioCDContents

isValid Returns true if the table of contents is valid.

Boolean isValid() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
-----------------------------	-----------------	----------------	-------------------

numberOfTracks

Returns the number of tracks.

unsigned long numberOfTracks() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

owner Returns the owner of the track at the given cursor. This value cannot be set because it is set by the manufacturer of the CD.

unsigned long owner(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

serialNumber Returns the serial number of the track at the given cursor. This value cannot be set because it is set by the manufacturer of the CD.

unsigned long serialNumber(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

startOfTrack Returns the starting time (position) of the track.

IMMTime startOfTrack(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

track Returns the track number at the given cursor.

unsigned long track(const Cursor& cursor) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

IMMAudioCDContents

IBase		
asString	messageText	version

Protected Functions

Constructors

You can construct, copy, and destruct objects of this class.

Use this function to create an IMMAudioCDContents.

IMMAudioCDContents

1	IMMAudioCDContents();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This creates an invalid table of contents.

2	IMMAudioCDContents(void* newContents, const IString& identifier, unsigned long tracks);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This function takes the following parameters:

- newContents* A pointer to a list of table of contents structures created by the operating system.
- identifier* The unique disc identifier generated by IMMAudioCD.
- tracks* The number of tracks to be in the playback list for the table of contents. IMMAudioCDContents creates a default playback list for the tracks.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IMMAudioCDContents contains the following nested classes:
IMMAudioCDContents::Cursor (see page 489)



IMMAudioCDContents::Cursor


Derivation IBase
IVBase
IMMAudioCDContents::Cursor

Inherited By None.

Header File immcdda.hpp

Members	Member	Page	Member	Page
	Constructor	489	operator =	490
	asIndex	490	setToFirst	491
	Cursor	489	setToIndex	491
	invalidate	490	setToLast	491
	isValid	490	setToNext	491
	operator ++	490	setToPrevious	491
	operator --	491	~Cursor	490

The IMMAudioCDContents::Cursor class creates and manages the cursor support of the table of contents. You can use a cursor to traverse through the list of tracks.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct, copy, and destruct objects of this class.

Cursor Creates an IMMAudioCDContents::Cursor.

1	Cursor(const IMMAudioCDContents& contents);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

This function creates a cursor on the passed in table of contents.

2	Cursor();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

This function creates an invalid cursor object. You have to set this cursor object equal to a valid cursor in order to use it.

IMMAudioCDContents::Cursor

3	<code>Cursor(const Cursor& cursor);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

This function is the copy constructor. It takes another cursor and sets this cursor to be the same as the passed in cursor.

operator = Sets this cursor to be the same as the passed in cursor.

<code>Cursor&</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>operator =(const Cursor& newCursor);</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

~Cursor

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~Cursor();</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

Cursor Validation and Cursor Movement

Use these members to change the cursor position in the list and to check the validity of the current position.

asIndex Returns the current index.

<code>unsigned long</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>asIndex() const;</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

invalidate Flags this cursor as invalid.

<code>virtual Cursor&</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>invalidate();</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

isValid Returns true if the cursor is valid.

<code>Boolean</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>isValid() const;</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

operator ++ Points to the next item in the table of contents. If no more items exist, this invalidates the cursor.

<code>virtual Cursor&</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>operator ++();</code>		<i>Y</i>	<i>Y</i>	<i>N</i>

IMMAudioCDContents::Cursor

operator -- Points to the previous item in the table of contents. If no previous items exist, this invalidates the cursor.

virtual Cursor& operator --();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToFirst Points to the table of contents item and validates the cursor.

virtual Cursor& setToFirst();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToIndex Points to the item with the given 1-based index and validates the cursor.

virtual Cursor& setToIndex(unsigned long toIndex);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToLast Points to the last table of contents item and validates the cursor.

virtual Cursor& setToLast();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToNext Points to the next item in the table of contents. If no more items exist, this invalidates the cursor.

virtual Cursor& setToNext();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToPrevious

Points to the previous item in the table of contents. If no previous items exist, this invalidates the cursor.

virtual Cursor& setToPrevious();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IMMAudioCDContents::Cursor

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMCDXA

Derivation

- IBase
- IVBase
- INotifier
- IStandardNotifier
- IMMDevice
- IMMPlayableDevice
- IMMRemovableMedia
- IMMCDXA

Inherited By None.

Header File immcdxa.hpp

Members	Member	Page	Member	Page
	Constructor	493	upc	494
	isOpenStringValid	496	~IMMCDXA	494

The IMMCDXA class provides behavior for devices that support CD-ROM Extended Architecture (CD-XA) discs. *CD-XA* refers to a storage format that accommodates interleaved storage of audio, video, and standard file data.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMCDXA

IMMCDXA(Boolean openNow = true, unsigned long instance = 0, Boolean openShareable = true);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

You can construct an IMMCDXA from the following:

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.

IMMCDXA

- instance

You can provide your own instance number instead of letting IMMCDXA generate one.
- openShareable

If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

~IMMCDXA

virtual
~IMMCDXA();

Win

PM

Motif

N

Y

N

UPC

Use these members to query the universal product code (upc) for the disc.

- upc

Returns the disc's universal product code (serial number). If the disc does not contain a upc, a null IString is returned.

IString
upc(CallType call = wait) const;

Win

PM

Motif

N

Y

N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

Inherited Public Functions

IMMRemovableMedia		
closeDoor	isMediaPresent	openDoor
enableNotification	lockDoor	unlockDoor

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame

IMMCDXA

IMMPlayableDevice		
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IMMCDXA

Protected Functions

Constructors

You can construct and destruct objects of this class.

IMMCDXA

IMMCDXA(unsigned long deviceIdentifier,
 const IString& newAlias = IString ());

WinPMMotif
N Y N

A derived class can construct an IMMCDXA from the following:

- deviceIdentifier* The value the system uses to identify the device.
- newAlias* The name you can use to associate a string to the device. Optional.

Implementation

These members check if the passed-in string is in the correct format to open the current device.

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

virtual Boolean
 isOpenStringValid(const IString& deviceName) const;

WinPMMotif
N Y N

Inherited Protected Functions

IMMRemovableMedia		
notificationHandler	setNotificationHandler	

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IMMCDXA

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMRemovableMedia		
mediaLoadedId		

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMMConfigurableAudio

IMMConfigurableAudio

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> INotifier
    INotifier --> IStandardNotifier
    IStandardNotifier --> IMMDevice
    IMMDevice --> IMMPlayableDevice
    IMMPlayableDevice --> IMMFileMedia
    IMMFileMedia --> IMMRecordable
    IMMRecordable --> IMMConfigurableAudio
```

Inherited By

IMMDigitalVideo
IMMWaveAudio

Header File

immaud.hpp

Members

Member	Page	Member	Page
Constructor	506	setBytesPerSecond	502
bitsPerSample	499	setChannels	502
blockAlignment	499	setFormat	503
bytesPerSecond	499	setSamplesPerSecond	503
channels	500	setVolume	504
format	500	translateAudioFlag	507
samplesPerSecond	500	volume	504
setBitsPerSample	501	~IMMConfigurableAudio	504
setBlockAlignment	501		

The IMMConfigurableAudio class is the base class for all devices that support audio. It provides functions for setting and querying the audio formats, speed, and audio channels.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Audio Attributes

Use these members to query and set the audio attributes for the current device.

IMMConfigurableAudio

bitsPerSample

Returns the number of bits-per-sample. This is the number of bits of data used to represent each sample of each channel. The standard values are 4, 8, and 16.

```
unsigned long  
bitsPerSample( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

blockAlignment

Returns the block alignment of data in bytes. The system processes a multiple of block-aligned bytes of data at a time. For example, if you have a 24-byte buffer, then you can only set a block alignment of 2, 3, 4, 6, 8, or 12.

```
unsigned long  
blockAlignment( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

bytesPerSecond

Returns the average number of bytes-per-second played or recorded. This is the rate the data is transferred to the hardware.

```
unsigned long  
bytesPerSecond( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMConfigurableAudio

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

channels Returns the number of audio channels set. For example, mono is 1; stereo is 2.

```
unsigned long  
channels( CallType call = wait ) const;
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

format Returns the interpretation of the audio format. See IMMAudioBuffer::Format (p. 465) for more information.

```
IMMAudioBuffer::Format  
format( CallType call = wait ) const;
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

samplesPerSecond

Returns the number of samples-per-second played or recorded. This is the sampling rate, in kilohertz, that each channel should use. Standard values are 11025, 22050, and 44100.

IMMConfigurableAudio

```
unsigned long  
    samplesPerSecond( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

setBitsPerSample

Sets the bits-per-sample to be played or recorded. This is the number of bits of data used to represent each sample of each channel. The standard values are 4, 8, and 16. You must set an audio format using setFormat (p. 503) before calling this function. The file is saved in this format.

```
virtual IMMConfigurableAudio&  
    setBitsPerSample( unsigned long bitsPerSample,  
                      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.
IAccessError	A audio format has not been set before calling this function.

setBlockAlignment

Sets the block alignment of the data. The system processes a multiple of block-aligned bytes of data at a time. For example, if you have a 24-byte buffer, then you can only set a block alignment of 2, 3, 4, 6, 8, or 12. You must set an audio format using setFormat (p. 503) before calling this function. The file is saved in this format.

```
virtual IMMConfigurableAudio&  
    setBlockAlignment( unsigned long alignment,  
                      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMConfigurableAudio

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.
IAccessError	An audio format has not been set before calling this function.

setBytesPerSecond

Sets the average bytes-per-second to be played or recorded. This is the rate the data is transferred to the hardware. You must set an audio format using setFormat (p. 503) before calling this function. The file is saved in this format.

```
virtual IMMConfigurableAudio&
    setBytesPerSecond( unsigned long averageBytes,
                      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.
IAccessError	An audio format has not been set before calling this function.

setChannels Sets the number of audio channels for playing and recording. Monaural is 1; stereo is 2. You must set an audio format using setFormat (p. 503) before calling this function. The file is saved in this format.

```
virtual IMMConfigurableAudio&
    setChannels( unsigned long channels,
                CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMConfigurableAudio

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.
IAccessError	An audio format has not been set before calling this function.

setFormat

Sets the audio format for playing and recording. See IMMAudioBuffer::Format (p. 465) for more information. The file is saved in this format.

```
virtual IMMConfigurableAudio&
    setFormat( IMMAudioBuffer::Format tag = IMMAudioBuffer::pcm, Win PM Motif
               CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.

setSamplesPerSecond

Sets the sampling rate for playing and recording. This is the sampling rate, in kilohertz, which each channel should use. Standard values are 11025, 22050, and 44100. You must set an audio format using setFormat (p. 503) before calling this function. The file is saved in this format.

```
virtual IMMConfigurableAudio&
    setSamplesPerSecond( unsigned long samplesPerSecond, Win PM Motif
                        CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.

IMMConfigurableAudio

Exceptions	
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The value is out of range.
IAccessError	An audio format has not been set before calling this function.

setVolume Sets the volume of the audio channel for the device to a percent (from 0% to 100%) of the maximum audio level.

```
virtual IMMConfigurableAudio&
setVolume(
    unsigned long volume,
    AudioChannel channel = all,
    const IMMILLISECONDTIME& over = IMMILLISECONDTIME ( ),
    CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



This member is overridden in this derived class for specific operating system behavior.

volume Returns the volume of the audio channel for the device as a percent (from 0% to 100%) of the maximum audio level.

```
virtual unsigned long
volume( AudioChannel channel = left,
        CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



This member is overridden in this derived class for specific operating system behavior.

Constructors

Derived classes can use these members to create objects of this class.

~IMMConfigurableAudio

```
virtual
~IMMConfigurableAudio();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMRecordable		
canRedo	cut	redo
canUndo	deleteSelection	save

IMMConfigurableAudio

IMMRecordable		
copy	paste	saveAs
cueForRecording	record	undo

IMMFileMedia		
filename	load	open
isWriteable	loadOnThread	openOnThread

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

IMMConfigurableAudio

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

Derived classes can use these members to create objects of this class.

IMMConfigurableAudio

1

IMMConfigurableAudio(const IString& deviceName,
Boolean openNow = true,
unsigned long instance = 0,
Boolean openShareable = true);

Win

PM

Motif

Y

Y

N

A derived class can construct an IMMConfigurableAudio from the following:

- deviceName

You must specify what device you want to construct.
- openNow

If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.
- instance

You can provide your own instance number instead of letting IMMConfigurableAudio generate one.
- openShareable

If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

2

IMMConfigurableAudio(unsigned long deviceIdentifier,
const IString& newAlias = IString ());

Win

PM

Motif

Y

Y

N

IMMConfigurableAudio

A derived class can construct an IMMConfigurableAudio from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Implementation

These members translate the audio command to device-specific values.

translateAudioFlag

Allows derived classes to map the general audio commands to the device-specific values. For example, the digital video and the wave audio device both support setting the bytes-per-second, but the actual system values are different. By allowing the mapping of the wave audio values to the digital video values, the digital video player does not have to override all of the audio functions because they call this function.

```
virtual unsigned long  
    translateAudioFlag( unsigned long ) const;
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMFileMedia		
enableDataUpdate	open	openOnThread

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

IMMConfigurableAudio

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMCuePointEvent

Derivation

```

IBase
IVBase
IEvent
IMMCuePointEvent
  
```

Inherited By None.

Header File immevt.hpp

Members	Member	Page	Member	Page
	Constructor	509	userParameter	510
	device	510	~IMMCuePointEvent	509
	position	510		

The IMMCuePointEvent class is the class for cue point events. A cue point event gets generated whenever a device passes over one of its set cue points.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMCuePointEvent

Although you can construct objects of this class, typically IMMDeviceHandler::dispatchHandlerEvent (p. 552) creates objects of this class from an object of the class IEvent (Vol. II) or another IMMCuePointEvent object.

```
IMMCuePointEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMCuePointEvent

```
virtual
~IMMCuePointEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMCuePointEvent

Event Information

Use these members to return the position, device, and user parameter for the device that generated this event.

device Returns a pointer to the device that played or recorded over a cue point.

```
IMMDevice*  
device() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

position Returns the cue point position.

```
IMMTime  
position() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

userParameter

Returns the user parameter that was set at the time of this event. See `IMMDevice::setUserParameter` (p. 537) for more information.

```
unsigned short  
userParameter() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

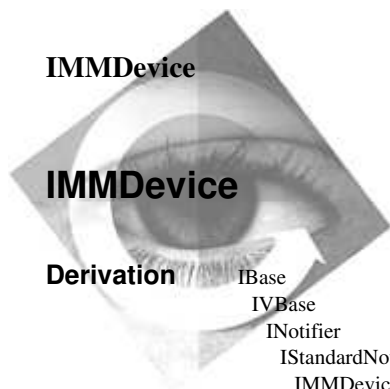
IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMDevice

IMMDevice

Derivation

IBase
IVBase
INotifier
IStandardNotifier
IMMDevice

Inherited By

IMMAmpMixer
IMMPlayableDevice

Header File

immdev.hpp

Members

Member	Page	Member	Page
Constructor	533	isOpenStringValid	535
acquire	528	itemCapability	535
aliasName	523	itemStatus	536
allDevices	539	lastError	534
ampMixer	540	microphone	540
animation	540	mixerControlValues	532
audioCD	540	mode	525
audioTape	540	monitor	541
cdxa	540	notificationHandler	535
close	526	open	526
commandNotifyId	542	openOnThread	527
connectedDeviceId	521	openStatus	539
cuePointId	542	other	541
dat	540	overlay	541
deletePendingEvents	525	passDeviceId	542
description	524	positionChangeId	543
deviceEventId	542	prerollTime	528
deviceId	523	prerollType	528
deviceName	524	release	529
deviceType	524	requiresFiles	516
deviceWindow	535	sendCommand	536
digitalVideo	540	sequencer	541
disableAudio	513	setCloseOnDestroy	527
disableConnector	521	setLastError	534
enableAudio	514	setMixerControlValues	532
enableConnector	522	setNotificationHandler	535
enableNotification	525	setOpenStatus	539
headphone	540	setPassDeviceRequested	537
isAcquired	529	setSpeedFormat	530
isAudioEnabled	514	setSystemMixerHandle	533
isCloseOnDestroy	526	setTimeFormat	530
isConnectionSupported	522	setUserParameter	537
isConnectorEnabled	523	setVolume	515
isOpen	526	speaker	541

IMMDevice

Member	Page
speedFormat	531
supportsAudio	516
supportsCommand	516
supportsDigitalTransfer	517
supportsDisableEject	517
supportsEject	517
supportsPlay	518
supportsRecord	518
supportsRecordInsertion	519
supportsSave	519
supportsStreaming	519
supportsVideo	520
supportsVolumeAdjustment	520
systemMixerHandle	533
timeFormat	531
userParameter	537
videoDisc	541
videoTape	541
volume	515
wasPassDeviceRequested	537
waveAudio	541
~IMMDevice	523

The IMMDevice class is the base device class and provides behavior common to all devices. IMMDevice contains the device ID, alias, and similar information. There are numerous functions for querying the capabilities of the device. Also, there are functions for opening and closing the device, changing the speed and time formats, and for changing the audio.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Audio

Use these members to manipulate the audio for the device. You can turn the audio off and on, and you can change and query the volume.

disableAudio Turns off the audio for the device. This audio change can be delayed based on the passed in vectored delay time (think of this the same as fade out on your car stereo). Based on this delay time, the audio is slowly turned off. The default is no delay.

<pre>virtual IMMDevice& disableAudio(AudioChannel channel = all, const IMMILLISECONDTIME& over = IMMILLISECONDTIME (), CallType call = wait);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

IMMDevice

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

enableAudio Turns on or turns off the audio for the device. This audio change can be delayed based on the passed in vectored delay (think of this the same as fade in on your car stereo). Based on this delay time, the audio is slowly turned on. The default is no delay.

```
virtual IMMDevice&
enableAudio(
    Boolean enable = true,
    AudioChannel channel = all,
    const IMMILLISECONDTIME& over = IMMILLISECONDTIME ( ),
    CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

isAudioEnabled

Returns true if the audio for the passed in channel is turned on.

```
Boolean
isAudioEnabled( AudioChannel channel = all,
    CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

IMMDevice

setVolume Sets the volume of the audio channel for the device to a percent (from 0% to 100%) of the maximum audio level.

```
virtual IMMDevice&
    setVolume(
        unsigned long volume,
        AudioChannel channel = all,
        const IMMILLISECONDTIME& over = IMMILLISECONDTIME ( ),
        CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

PM This volume change can be delayed based on the passed-in vectored delay time (think of this the same as fade in or fade out on your car stereo). Based on this delay time, the volume is slowly changed to the new volume. The default is no delay.

In order to affect the volume of CD audio when using a cdStream, you must create an IMMAMP_Mixer to control the volume.

Win For Win32, the vectored delay time is not supported.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

volume Returns the volume setting for the passed-in audio channel. This value is in the range of 0 to 100.

```
virtual unsigned long
    volume( AudioChannel channel = left,
            CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

IMMDevice

Capabilities

Use these members to determine the capabilities of the device. All devices should respond to these inquiries.

requiresFiles Returns true if the device requires the use of files. An example of a device that requires files is the wave audio device, and an example of a device that does not require files is the amplifier-mixer device.

```
Boolean  
requiresFiles( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsAudio

Returns true if the device has support for audio playback.

```
Boolean  
supportsAudio( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsCommand

Returns true if the device supports the passed-in command.

```
Boolean  
supportsCommand( IMMNotifyEvent::Command command,  
                 CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMDevice

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsDigitalTransfer

Returns true if the device can internally process digital data, for example a digital-to-analog converter on a CD player.

Boolean	Win	PM	Motif
supportsDigitalTransfer(CallType call = wait) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsDisableEject

Returns true if the device can disable the manual ejection of the media.

Boolean	Win	PM	Motif
supportsDisableEject(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

supportsEject

Returns true if the device can eject the media.

IMMDevice

Boolean	Win	PM	Motif
supportsEject(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsPlay Returns true if the device can play.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
supportsPlay(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsRecord

Returns true if the device supports recording. This does not necessarily mean it also supports the ability to save the data you just recorded. See `supportsSave` (p. 519) for more information.

Boolean	Win	PM	Motif
supportsRecord(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

IMMDevice

supportsRecordInsertion

Returns true if the device supports insertion of data while recording. For example, you would use this function if you just recorded a sound file and then wanted to record your voice saying "Hello" in the middle of it without overlaying any of the recorded sound.

Boolean	Win	PM	Motif
supportsRecordInsertion(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsSave

Returns true if the device can save data to some type of media (for example, a file). An example of a device that supports saving is the wave audio device, and an example of a device that does not support saving is the audio CD device.

Boolean	Win	PM	Motif
supportsSave(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsStreaming

Returns true if the device can continuously transfer digital data to or from another device. The source or destination of the data is determined by the device's connections.

Boolean	Win	PM	Motif
supportsStreaming(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

IMMDevice

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsVideo

Returns true if the device has support for video playback.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
supportsVideo(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IIllegalRequest	The device must be in the open state before calling this function.

supportsVolumeAdjustment

Returns true if the device supports software control of the volume. An example of a device that supports volume adjustment is the wave audio device, and an example of a device that does not support audio adjustment is the video disc device.

Boolean	Win	PM	Motif
supportsVolumeAdjustment(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Connector Type

Use these members to query and manipulate the connectors for the device.

IMMDevice

Connectors give devices the ability to connect to other devices. For those of you who have VCRs or stereos, you probably are familiar with those RCA connection cables that you use to connect different stereo and video components. You normally connect one end of the cable to a line in (such as video in on your TV) and connect the other end to the line out (such as video out on your VCR). Think of these software connections and connectors as those cables and connections.

For example, suppose you want to record your favorite song from a cassette as a sound file on your computer, but you do not have a microphone. You do, however, have a RCA cable connected to your stereo. You could connect one end of the cable to your cassette player and the other to the line in of your sound card. By default, the wave audio device would record from the microphone jack on the sound card because it is initially connected to the microphone connector. To record from the sound card's line in, you would have to make your wave audio device record from the line in connector on the sound card. To do this, you would enable the line in connector for your audio device by calling *enableConnector(IMMDevice::lineIn)*; on your audio device. Because the audio device only supports one input connection to be enabled at a given time, it automatically disables the microphone connector. So, to create the sound file, all you would have to do is to press play on the cassette player and call the record function on the wave audio device.

connectedDeviceId

Returns the ID of the device that is connected to the identified connector of this device.

```
unsigned long  
connectedDeviceId( ConnectorType type,  
                  CallType call = wait ) const;
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

disableConnector

Turns off the connector specified by *type* for this device.

```
virtual IMMDevice&  
disableConnector( ConnectorType type,  
                  CallType call = wait );
```

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>I</i>	<i>Y</i>	<i>N</i>

IMMDevice

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	The connector type is not supported by this device.

enableConnector

Turns on or turns off the connector specified by *type* for this device.

```
virtual IMMDevice&  
    enableConnector( ConnectorType type,  
                    Boolean enable = true,  
                    CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	The connector type is not supported by this device.

isConnectionSupported

Returns true if the connector is valid for this device.

```
Boolean  
    isConnectionSupported( ConnectorType type,  
                          CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

IMMDevice

isConnectorEnabled

Returns true if the connector is enabled for this device.

```
Boolean  
isConnectorEnabled( ConnectorType type,  
                   CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>I</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Constructors

Derived classes can use these members to create objects of this class.

~IMMDevice

```
virtual  
~IMMDevice();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Device Information

Use these members to query information about the device and its state.

aliasName Returns the alias associated with the device during the open procedure. This function is can return an empty string because a device can be constructed solely from a device ID.

```
IStrng  
aliasName() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

deviceId Returns the ID of the current device. This value is used by the system to identify the device. In the construction of this object, a device identifier is passed in.

```
unsigned long  
deviceId() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMDevice

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

deviceName Returns the name of the device depending on the class of the object. For example, some devices can be opened with a file name (song.mid), but other devices, such as IMMAudioCD, are opened with a specific device name (such as CDAUDIO01).

HRESULT	<u>Win</u>	<u>PM</u>	<u>Motif</u>
deviceName() const;	Y	Y	N

Device Type and Description

Use these members to query the type and description of the device.

description Returns the description of the hardware associated with this device.

HRESULT	<u>Win</u>	<u>PM</u>	<u>Motif</u>
description(CallType call = wait) const;	Y	Y	N

deviceType Returns the type of the device.

HRESULT	<u>Win</u>	<u>PM</u>	<u>Motif</u>
deviceType(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Event Dispatching

Event-dispatching members interact with the notification handlers for this device.

IMMDevice

enableNotification

Flags this object as being ready to accept notifications. A Notifier, like IMMDevice, does not send notifications to observer objects until it is enabled.

```
virtual IMMDevice&  
    enableNotification( Boolean enabled = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Event Type and DeletePendingEvents

Use these members to specify the type of pending events that are to be deleted.

deletePendingEvents

Removes all of the passed-in event types from the queue.

```
virtual IMMDevice&  
    deletePendingEvents( EventType event = allEvents );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Mode

Use these members to query the current mode of the device.

mode

Returns the current mode of the device.

```
virtual Mode  
    mode( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

IMMDevice

Opening and Closing

Use these members to query if the device is opened, to open or close the device, and to direct the device to close itself when the device object is destructed.

close Closes a device.

```
virtual IMMDevice&  
    close( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

isCloseOnDestroy

Returns true if the device closes itself when the device object is deleted.

```
Boolean  
    isCloseOnDestroy() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isOpen Returns true if the device is opened. In general, open a device before calling multimedia functions.

```
Boolean  
    isOpen() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.

open Opens the device or file based on the passed-in string. The default string uses the deviceName. Note, if *call* is wait (p. 543), then the windowing system will be tied up until the file or device is opened. If you do not want to tie up the windowing system and you do not want the open function to return until the file or device is opened, then use the openOnThread (Vol. II) function. Otherwise, you could set *call* to nowait (p. 543), causing open to return immediately. When the open finishes, an

IMMDevice

INotificationEvent (Vol. I) is sent to any observers that are attached to this device, notifying them that the open is finished.

You must have opened the device before you try to acquire (p. 528) access to it. When a device is open, it notifies the operating system that you want to have the use of the hardware at sometime in the future, and notifies the operating system that it can share the device. When you acquire access, you are requesting that the system provide you access to the hardware now. Then, based on the type of ShareMode (p. 546) that you requested, it gives you access to the hardware.

```
virtual IMMDevice&
  open( const IString& fileOrDevice = IString ( ),           Win PM Motif
        Boolean shareable = true,                          Y   Y   N
        CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid;
IAccessError	An internal driver error occurred.
IAccessError	The device is locked.
IAccessError	A hardware error occurred.

openOnThread

Opens the file or device by creating a thread to do the actual opening. The default string uses the deviceName. By doing the opening this way, it does not tie up the windowing system. This function does not return until the file or device is opened.

```
virtual IMMDevice&
  openOnThread( const IString& fileOrDevice = IString ( ),   Win PM Motif
               Boolean shareable = true,                     Y   Y   N
               CallType call = wait );
```



The Win32 MCI does not support threads, so this function behaves like open() in this environment.

Exceptions	
IAccessError	The device identifier is not valid;
IAccessError	An internal driver error occurred.
IAccessError	The device is locked.
IAccessError	A hardware error occurred.

setCloseOnDestroy

Closes the device when this object is destroyed. The default is to close the device.

IMMDevice

```
IMMDevice&  
    setCloseOnDestroy( Boolean close = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Preroll Type

Use these members to prepare a device to begin a playback or recording function with minimal delay. These members set and query the amount of preroll time.

prerollTime Returns the amount of preroll time in IMMTime (p. 680). If the returned time object is invalid, then the preroll time is not bounded.

```
IMMTime  
    prerollTime( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

prerollType Returns the preroll characteristics of the device.

```
PrerollType  
    prerollType( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

Share Mode and Acquiring

Use these members to specify the three ways that the operating system can share access to the device and to query, acquire, and release access to the device.

acquire Requests assignment of (access to) the device. If queued is true, the request is queued and executed as soon as the device resources are available. If the request can be satisfied immediately, then it is not queued.

IMMDevice

You must have opened the device before you try to acquire access to it. When a device is open, it notifies the operating system that you want to have the use of the hardware at some time in the future and notifies the operating system that it can share the device. When you acquire access, you are requesting that the system provide you access to the hardware now. Then, based on the type of ShareMode (p. 546) that you requested, it gives you access to the hardware.

```
virtual IMMDevice&
    acquire( ShareMode acquire = shareable,
             Boolean queuedForResources = false,
             CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

isAcquired Returns true if the device is ready. If a device is not ready, then access to it needs to be acquired using acquire (p. 528).

```
Boolean
    isAcquired( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

release Releases exclusive use of the device resources.

```
virtual IMMDevice&
    release( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMDevice

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Speed and Time

Use these members to set and query the current formats for speed and time for the device.

setSpeedFormat

Sets the speed format for the device. See `IMMSpeed` (p. 677) for more information on the different speed formats.

```
virtual IMMDevice&
    setSpeedFormat( IMMSpeed::Format format,
                   CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



For Win32, the only supported format is `framesPerSecond`.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

setTimeFormat

Sets the time format for the device. See `IMMTime::Format` (p. 687) enumeration for more information on the different time formats.

```
virtual IMMDevice&
    setTimeFormat( IMMTime::Format format,
                  CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.

IMMDevice

Exceptions	
InvalidRequest	The device must be in the open state before calling this function.

speedFormat Returns the currently set speed format for the device. See IMMSpeed (p. 677) for more information on the different speed formats.

IMMSpeed::Format
speedFormat(CallType call = wait) const; Win PM Motif
Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

timeFormat Returns the currently set time format for the device. See the Format (p. 687) enumeration for more information on the different time formats.

IMMTime::Format
timeFormat(CallType call = wait) const; Win PM Motif
Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Inherited Public Functions

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IMMDevice

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Audio

Use these members to manipulate the audio for the device. You can turn the audio off and on, and you can change and query the volume.

mixerControlValues

Returns one or more values associated with a mixer control.

1

virtual Boolean
 mixerControlValues(unsigned long dwComponentType,
 unsigned long dwControlType,
 unsigned short* usDetails,
 Boolean uniform = true,
 unsigned long multipleItems = 0,
 unsigned long hmx = NULL);

Win
Y

PM
N

Motif
N

This function returns details for a mixer control that has two states, on and off (such as a mute button).

2

virtual Boolean
 mixerControlValues(unsigned long dwComponentType,
 unsigned long dwControlType,
 unsigned long* ulDetails,
 Boolean uniform = true,
 unsigned long multipleItems = 0,
 unsigned long hmx = NULL);

Win
Y

PM
N

Motif
N

This function returns details for a mixer control that has many states, such as a volume control. The values returned are a percentage of a maximum value, in the range from 0 to 100.

setMixerControlValues

Sets one or more values associated with a mixer control.

IMMDevice

1	virtual Boolean setMixerControlValues(unsigned long dwComponentType, unsigned long dwControlType, unsigned short* usDetails, Boolean uniform = true, unsigned long multipleItems = 0, unsigned long hmx = NULL);	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>N</i>	<i>N</i>						

This function sets details for a mixer control that has two states, on and off (such as a mute button).

2	virtual Boolean setMixerControlValues(unsigned long dwComponentType, unsigned long dwControlType, unsigned long* ulDetails, Boolean uniform = true, unsigned long multipleItems = 0, unsigned long hmx = NULL);	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>N</i>	<i>N</i>						

This function sets details for a mixer control that has many states, such as a volume control. The values should be a percentage of a maximum value, in the range from 0 to 100. Any value greater than 100 is treated as 100.

setSystemMixerHandle

Sets the system mixer handle for this device.

IMMDevice& setSystemMixerHandle(unsigned long hmx);	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

systemMixerHandle

Returns the currently set system mixer handle for this device. If one does not exist, this function will open a system mixer handle and set it for this device before returning.

unsigned long systemMixerHandle() const;	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Constructors

Derived classes can use these members to create objects of this class.

IMMDevice

IMMDevice

1	<code>IMMDevice(const IString& fileOrDevice);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMDevice from the following:

fileOrDevice

This is the name of the device. This name is operating-system-specific. For example the name of the amplifier-mixer device in the OS/2 operating system is AMPMIX.

2	<code>IMMDevice(unsigned long deviceIdentifier, const IString& newAlias = IString ());</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMDevice from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Error Information

Use these members to set and query an error value.

lastError Returns the return code from the last call. This return code could be 0 because the value is reset with each call. If an error code needs to be captured, this function should be called directly after the error-causing function because each function overwrites this value.

<code>unsigned long lastError() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

PM Because the system multimedia architecture does not keep track of the return value of the last function call, this function does that. It allows all of the functions to throw exceptions where they are meaningful.

setLastError Sets the return code of the last low-level function call.

<code>IMMDevice& setLastError(unsigned long errorId);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

PM Because the system multimedia architecture does not keep track of the return value of the last function call, this function does that. It allows all of the functions to throw exceptions where they are meaningful.

Event Dispatching

Event-dispatching members interact with the notification handlers for this device.

notificationHandler

Returns a pointer to the notification handler

IMMDeviceNotifyHandler*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
notificationHandler() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setNotificationHandler

Sets the notification handler to the passed value.

IMMDevice&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setNotificationHandler(IMMDeviceNotifyHandler* notifyHandler);	<i>Y</i>	<i>Y</i>	<i>N</i>

Implementation

Use these members to access the low-level functions and to set and query a user parameter that can be passed on function calls.

deviceWindow

Returns the window object that the device uses to communicate to the operating system.

IMMMessageWindow&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
deviceWindow() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isOpenStringValid(const IString& deviceName) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

itemCapability

Gets the device capabilities for the item. This is similar to using the devQueryCaps system call on the OS/2 operating system.

IMMDevice

```
unsigned long  
    itemCapability( unsigned long item,  
                    CallType call ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

itemStatus Checks the status of the item.

```
unsigned long  
    itemStatus( unsigned long item,  
                unsigned long value = 0,  
                CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

sendCommand

Sends a command string to the device. This returns the resulting string from the call.

1 IString
 sendCommand(const IString& command,
 unsigned short userParm);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

The passed in *userParameter* is in the notification message and does not replace the value set with setUserParameter (p. 537).

2 IString
 sendCommand(const IString& command);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

The currently set userParameter (p. 537) is in the notification message.

IMMDevice

3	IMMDevice& sendCommand(unsigned short message, unsigned long param1, void* param2);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------	--	------------------------	-----------------------	--------------------------

The currently set *userParameter* (p. 537) is in the notification message.

4	IMMDevice& sendCommand(unsigned short message, unsigned long param1, void* param2, unsigned short userParm);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------	--	------------------------	-----------------------	--------------------------

The passed in *userParameter* is in the notification message and does not replace the value set with the *setUserParameter* (p. 537).

setPassDeviceRequested

Sets the pass device flag. Use this to notify the device object that the device object has received use of the actual hardware device. Until the device receives the first pass device event, the device is not initialized.

IMMDevice& setPassDeviceRequested(Boolean sawPassDevice);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	------------------------	-----------------------	--------------------------

setUserParameter

Sets the user parameter. This parameter is automatically passed in the notification messages for all but two of the device functions. The exceptions are the two *sendCommand* (p. 536) functions that take their own user parameter.

IMMDevice& setUserParameter(unsigned short userParm = 0);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	------------------------	-----------------------	--------------------------

userParameter

Returns the user parameter that is in all of the notification messages except for the two *sendCommand* (p. 536) functions that take their own user parameter.

unsigned short userParameter() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	------------------------	-----------------------	--------------------------

wasPassDeviceRequested

Returns true if it has already processed a *passDevice* event. Use this to notify the device object that the device object has received use of the actual hardware device. Until the device receives the first pass device event, the device is not initialized.

IMMDevice

Boolean
wasPassDeviceRequested() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Opening and Closing

Use these members to query if the device is opened, to open or close the device, and to direct the device to close itself when the device object is destructed.

open Opens the device.

Note: Do not use open for devices that take a while to open because it ties up the windowing system till the device is opened. Use the openOnThread function for this case.

```
virtual IMMDevice&  
open( unsigned long instanceNumber,  
      Boolean shareable = true,  
      CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid;
IAccessError	An internal driver error occurred.
IAccessError	The device is locked.
IAccessError	A hardware error occurred.

openOnThread Opens the device by creating a thread to do the actual opening. By doing the opening this way, it will not tie up the windowing system.

```
virtual IMMDevice&  
openOnThread( unsigned long instanceNumber,  
              Boolean shareable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The Win32 MCI does not support threads, so this function behaves like open() in this environment.

Exceptions	
IAccessError	The device identifier is not valid;
IAccessError	An internal driver error occurred.
IAccessError	The device is locked.
IAccessError	A hardware error occurred.

OpenStatus

Use these members to set and query the open state of the device.

openStatus Returns the result of the open attempt. This is used by the device to determine if the opening of the device has occurred yet and if it was successful. This is important because basically all of the multimedia function calls require the device to be opened first before you can make any calls.

OpenStatus	<u>Win</u>	<u>PM</u>	<u>Motif</u>
openStatus() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setOpenStatus

Sets the open flag. This is used by the device to determine if the opening of the device has occurred yet and if it was successful. This is important because basically all of the multimedia function calls require the device to be opened first before you can make any calls.

IMMDevice&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setOpenStatus(OpenStatus status);	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Public Data**Device Type and Description**

Use these members to query the type and description of the device.

allDevices A device type for representing all devices.

static const unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
allDevices;	<i>Y</i>	<i>Y</i>	<i>N</i>

IMMDevice

ampMixer	A device type for representing a combination amplifier-mixer that is used to control the characteristics of an audio signal from one or more audio sources.						
static const unsigned long ampMixer;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
animation	A device type for representing a device that uses a sequence of images to produce the effect of movement.						
static const unsigned long animation;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
audioCD	A device type for representing an audio CD device.						
static const unsigned long audioCD;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
audioTape	A device type for representing an audio tape recorder device.						
static const unsigned long audioTape;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
cdxa	A device type for representing a compact disc-extended architecture (CD-XA) device.						
static const unsigned long cdxa;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
dat	A device type for representing a digital audio tape device.						
static const unsigned long dat;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
digitalVideo	A device type for representing a digital video device.						
static const unsigned long digitalVideo;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
headphone	A device type for representing a headphone.						
static const unsigned long headphone;	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					
microphone	A device type for representing a microphone.						

IMMDevice

	static const unsigned long microphone;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
monitor	A device type for representing a device that can listen to (monitor) an audio and video source.			
	static const unsigned long monitor;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
other	A device type for representing devices other than the system-provided device types.			
	static const unsigned long other;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
overlay	A device type for representing a device that can overlay an image on top of the screen that does not disrupt what is displayed there.			
	static const unsigned long overlay;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
sequencer	A device type for representing a MIDI synthesizer device.			
	static const unsigned long sequencer;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
speaker	A device type for representing output to the speakers.			
	static const unsigned long speaker;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
videoDisc	A device type for representing a video disc player device.			
	static const unsigned long videoDisc;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
videoTape	A device type for representing a video tape device.			
	static const unsigned long videoTape;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
waveAudio	A device type for representing a device that uses digital audio data in the wave format.			

IMMDevice

```
static const unsigned long  
waveAudio;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Notification Event Descriptions

These INotificationId strings are used for all notifications that IMMDevice provides to its observers.

commandNotifyId

Provides a notification identifier to observers when one of the IMMDevice functions has been called with the notify flag. IMMDevice provides a pointer to an IMMNotifyEvent (p. 613) for the command in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const  
commandNotifyId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

cuePointId

Provides a notification identifier to observers when the device receives a cue point message. IMMDevice provides a pointer to an IMMCuePointEvent (p. 509) for the message in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const  
cuePointId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

deviceEventId

Provides a notification identifier to observers when the device receives a device message. IMMDevice provides a pointer to an IMMDeviceEvent (p. 547) for the message in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const  
deviceEventId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

passDeviceId

Provides a notification identifier to observers when the device receives a pass device message. IMMDevice provides a pointer to an IMMPassDeviceEvent (p. 617) for the message in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const  
passDeviceId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMDevice

positionChangeId

Provides a notification identifier to observers when the device receives a position change message. IMMDevice provides a pointer to an IMMPositionChangeEvent (p. 643) for the message in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const  
    positionChangeId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

AudioChannel

```
AudioChannel {  
    all,  
    left,  
    right  
};
```

Use this enumeration to specify audio channel identifiers.

CallType

```
CallType {  
    nowait = 0X00000001,  
    wait = 0x00000002  
};
```

This enumeration determines how the multimedia functions call the low level operating system functions. In both call types, a notification event is sent to all attached IObservers (Vol. I) when the command finishes processing. Most functions that result in a call to the operating system take this type as their last parameter. The following are valid values:

IMMDevice

nowait

The method is asynchronous and returns immediately (before the requested processing is complete). If you want to know when the command finishes processing, attach an observer to the device because it gets notified when the command finishes.

wait

The method is synchronous and all processing occurs before the function returns. Wait is not recommended for threads having a message queue that is processing messages for user interface windows. A long wait could tie up the message queue.

ConnectorType

```
ConnectorType {  
    midiStream = 0x00000001,    cdStream = 0x00000002,  
    waveStream = 0x00000003,    ampStream = 0x00000004,  
    xaStream = 0x00000005,      headphones = 0x00000006,  
    speakers = 0x00000007,      microphones = 0x00000008,  
    lineIn = 0x00000009,        lineOut = 0x0000000a,  
    videoIn = 0x0000000b,       videoOut = 0x0000000c,  
    phoneSet = 0x0000000d,      phoneLine = 0x0000000e,  
    audioIn = 0x0000000f,       audioOut = 0x00000010,  
    universal = 0x00000011  
};
```

Use the preceding enumerations to specify connection types. These are all of the types of connections that a device can support. Each device only supports a subsection of this list. For example, an audio CD device would not support xaStream or microphones.

Note: “Stream in” these enumerations means the digital transfer of the data for these devices.

EventType

```
EventType {  
    allEvents,                notifyEvent = 0x0500, passDeviceEvent,  
    positionChangeEvent,      cuePointEvent,    deviceEvent,  
    synchEvent  
};
```

Use the preceding enumerations to specify the type of events that are to be removed.

Mode

```
Mode {  
    notReady = 1, paused,      playing,      stopped,      recording,  
    seeking  
};
```

IMMDevice

An enumeration that denotes the current state of the device. The following are valid values:

notReady

A device is in this state when the device is not acquired.

paused

The device is paused.

playing

The device is playing.

stopped

The device is stopped.

recording

The device is recording.

seeking

The device is moving the current position to a new location.

OpenStatus

```
OpenStatus {  
    openNotAttempted,  
    openSuccessful,  
    openFailed  
};
```

An enumeration that denotes the different states for opening a device. The following are valid values:

openNotAttempted

You have not tried to open the device yet.

openSuccessful

The open succeeded.

openFailed

The open failed.

PrerollType

```
PrerollType {  
    variable = 1,  
    fixed,  
    none  
};
```

Use these enumerators to specify preroll characteristic identifiers. *Preroll* means preparing a device to begin a playback or recording function with minimal delay. The following are valid values:

IMMDevice

variable

The preroll time for the device is variable, meaning that you can change the preroll time value.

fixed

The preroll time for the device is fixed, meaning that you cannot change the preroll time value; it is fixed by the operating system.

none

The preroll time is not supported by the device.

ShareMode

```
ShareMode {  
    shareable,  
    isolatedExclusive = 0x00000100,  
    exclusive = 0x00000200  
};
```

Use these enumerators to specify share mode identifiers:

shareable

Allows the device to be shared by other applications.

isolatedExclusive

Inhibits other applications from acquiring use of the physical (or hardware) device until released by the owning application.

exclusive

Acquires the device for exclusive use without acquiring the entire device resource for exclusive use. This basically means that the operating system cannot make this device inactive.



IMMDeviceEvent


Derivation IBase
 IVBase
 IEvent
 IMMDeviceEvent

Inherited By None.

Header File immevt.hpp

Members	Member	Page	Member	Page
	Constructor	547	eventData	548
	device	548	~IMMDeviceEvent	547
	eventCode	548		

The IMMDeviceEvent class is the class for device events. These events get generated by devices for device-specific purposes.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMDeviceEvent

Although you can construct objects of this class, typically IMMDeviceHandler::dispatchHandlerEvent (p. 552) creates objects of this class from an object of the class IEvent (Vol. II) or another IMMDeviceEvent object.

IMMDeviceEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

~IMMDeviceEvent

virtual ~IMMDeviceEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IMMDeviceEvent

Event Information

Use these members to return the device, event code, and event data for the device that generated this event.

device Returns a pointer to the device that this event is from.

```
IMMDevice*  
device() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

eventCode Returns the device-specific event code.

```
unsigned short  
eventCode() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

eventData Returns the device-specific event data structure.

```
void*  
eventData() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMDeviceHandler

IMMDeviceHandler

Derivation


IBase
IVBase
IHandler
IMMDeviceHandler

Inherited By IMMRemovableMediaHandler

Header File immdvhdr.hpp

Members				
	Member	Page	Member	Page
	Constructor	550	passDevice	551
	cuePoint	551	positionChange	552
	deviceEvent	551	stopHandlingEventsFor	551
	dispatchHandlerEvent	552	~IMMDeviceHandler	551
	handleEventsFor	551		

The IMMDeviceHandler class is the base handler class for IMMDevices and provides behavior common to all devices. IMMDeviceHandler inherits from IHandler. It inherits dispatchHandlerEvent virtual function from IHandler to encapsulate the multimedia interface. This function does four things. First, it determines what message has been received and converts it into the appropriate event. Second, it determines which device the event is for. Third, it calls the virtual function for the event. Fourth, it goes through the list of added IObservers for the device and calls the IObserved::dispatchNotificationEvent with the event.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMDeviceHandler

IMMDeviceHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IMMDeviceHandler

~IMMDeviceHandler

virtual ~IMMDeviceHandler();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---------------------------------	------------------------	-----------------------	--------------------------

Device Attachment

Use these members to attach and detach the handler object to and from a given device.

handleEventsFor

Attaches the handler to the specified IMMDevice (p. 512) object.

virtual IMMDeviceHandler& handleEventsFor(IMMDevice* device);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

stopHandlingEventsFor

Detaches the handler from the specified IMMDevice (p. 512) object.

virtual IMMDeviceHandler& stopHandlingEventsFor(IMMDevice* device);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

Event Processing

Derived classes must supply these members to process a device, position change, pass, or cue point event.

cuePoint Implemented by derived classes to process cuePoint events.

virtual Boolean cuePoint(const IMMCuePointEvent& event);	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

deviceEvent Implemented by derived classes to process device events.

virtual Boolean deviceEvent(const IMMDeviceEvent& event);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

passDevice Implemented by derived classes to process pass device events.

virtual Boolean passDevice(const IMMPassDeviceEvent& event);	<u>Win</u> <i>I</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

IMMDeviceHandler

positionChange

Implemented by derived classes to process position change events.

```
virtual Boolean  
    positionChange( const IMMPositionChangeEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process. If it is, these members call the virtual function used to process the event.

dispatchHandlerEvent

This function does four things. First, it determines what message has been received and converts it into the appropriate event. Second, it determines which device the event is for. Third, it calls the virtual function for the event. Fourth, it goes through the list of added IObservers for the device and calls the `IObserver::dispatchNotificationEvent` with the event.

A given handler should return true if the event should not be dispatched to other handlers. In such cases, a result can be placed in the specified `IEvent` (Vol. II) object.

IMMDeviceHandler

```
virtual Boolean  
    dispatchHandlerEvent( IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMDeviceNotifyHandler

IMMDeviceNotifyHandler




Inherited By IMMRemovableMediaNotifyHandler

Header File immdevnh.hpp

Members	Member	Page	Member	Page
	Constructor	554	setNotificationHandler	556
	dispatchHandlerEvent	555	~IMMDeviceNotifyHandler	555
	notificationHandler	555		

The IMMDeviceNotifyHandler class processes events for all IMMDevices.

This class is designed to handle events that require the IMMDevice to generate a notification. If notifications are enabled for this class, a notification is generated and sent to all observers when the proper conditions for the specific notification exist.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMDeviceNotifyHandler

This is the default constructor and accepts no parameters.

IMMDeviceNotifyHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IMMDeviceNotifyHandler

~IMMDeviceNotifyHandler

```
virtual  
~IMMDeviceNotifyHandler();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it.

dispatchHandlerEvent

Notifies observers if any of the following events are received:

- command notify event
- pass device event

```
virtual Boolean  
dispatchHandlerEvent( IEvent& anEvent );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

notificationHandler

Returns a pointer to the notification handler being used by the device.

IMMDeviceNotifyHandler

IMMDeviceNotifyHandler* notificationHandler() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

setNotificationHandler

Sets the notification handler to be used by the device.

IMMDeviceNotifyHandler& setNotificationHandler(IMMDeviceNotifyHandler* notifyHandler);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMDigitalVideo

Derivation	IBase
	IVBase
	INotifier
	IStandardNotifier
	IMMDevice
	IMMPlayableDevice
	IMMFileMedia
	IMMRecordable
	IMMConfigurableAudio
	IMMDigitalVideo

Inherited By None.

Header File immdigvd.hpp

Members				
	Member	Page	Member	Page
	Constructor	559	record	567
	destinationRectangle	568	refresh	569
	disableMonitoring	566	setDestination	570
	enableMonitoring	566	setMonitorWindow	567
	fastSpeed	563	setWindow	570
	fileNormalSpeed	563	slowSpeed	565
	handle	568	sourceRectangle	571
	isMonitoringEnabled	567	speed	565
	isOpenStringValid	576	supportsOverlayGraphics	558
	isPlayingForward	560	supportsReverse	558
	maximumSpeed	564	supportsSizing	558
	maximumWindows	569	supportsStretchToFit	559
	minimumSpeed	564	translateAudioFlag	576
	monitorHandle	567	useDefaultMonitorWindow	568
	normalSpeed	565	useDefaultWindow	571
	play	560	videoFileHeight	572
	playAt	561	videoFileName	572
	playDigital	576	videoFileWidth	572
	playFast	562	videoHeight	573
	playScan	562	videoWidth	573
	playSlow	563	~IMMDigitalVideo	560

The IMMDigitalVideo class provides behavior for digital software motion video. It supports playback of video in either a system-provided default window or a user-specified window. It provides functions for playing and controlling video in a default window or an application window. There are also functions for playing the video at different speeds.

IMMDigitalVideo

W_{32s}

The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Capabilities

Use these members to query the capabilities for the current device.

supportsOverlayGraphics

Returns true if the device supports the display by an application of overlay graphics in a video window.

Boolean

```
supportsOverlayGraphics( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsReverse

Returns true if the device can play in reverse.

Boolean

```
supportsReverse( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsSizing

Returns true if the device can independently stretch the horizontal and vertical dimensions of the image. This is different than `supportsStretchToFit` (p. 559), which forces both horizontal and vertical dimensions to be stretched in the same proportion.

IMMDigitalVideo

For example, if you stretch the horizontal dimension by 10%, then the vertical dimension must stretch by 10%.

Boolean Win PM Motif
supportsSizing(CallType call = wait) const; Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

supportsStretchToFit

Returns true if the device can stretch the video frames to fill the display rectangle. Thus, both the horizontal and vertical dimensions must be stretched by the same proportion. For example, if you stretch the horizontal dimension by 10%, then the vertical dimension must stretch by 10%. This is different than supportsSizing (p. 558), which allows both horizontal and vertical dimensions to be stretched independently of each other.

Boolean Win PM Motif
supportsStretchToFit(CallType call = wait) const; Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Constructors

You can construct and destruct objects of this class.

IMMDigitalVideo

IMMDigitalVideo(Boolean openNow = true,
unsigned long instance = 0,
Boolean openShareable = true); Win PM Motif
Y Y N

IMMDigitalVideo

You can construct an IMMDigitalVideo from the following:

- openNow* If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.
- instance* You can provide your own instance number instead of letting IMMDigitalVideo generate one.
- openShareable*
If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

~IMMDigitalVideo

```
virtual  
~IMMDigitalVideo();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Playback

Use these members to play a video at different rates.

isPlayingForward

Returns true if the play direction is forward or if the device is not playing.

```
Boolean  
isPlayingForward( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

play

Starts playing the device from the passed-in start position to the passed-in end position. If *from* is omitted, the device starts playing at the current position; if *to* is omitted, play stops at the end of the data. If *resumeIfPaused* is true and the device is paused, it resumes playback of the device; otherwise, it starts playback from the passed in start position.

IMMDigitalVideo

```
virtual IMMDigitalVideo&
play( const IMMTime& from = IMMTime ( ),
      const IMMTime& to = IMMTime ( ),
      Boolean resumeIfPaused = true,
      CallType call = nowait );
```

Win **PM** **Motif**
Y N N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There is no data present; possibly no file is loaded or no media is in the device.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the data.

playAt

Plays the video at the specified speed from the start position to the end position. If *from* is omitted, play starts at the current position; if *to* is omitted, play stops at the end of the video file.

```
IMMDigitalVideo&
playAt( const IMMSpeed& speed,
        const IMMTime& from = IMMTime ( ),
        const IMMTime& to = IMMTime ( ),
        CallType call = nowait );
```

Win **PM** **Motif**
Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the video file.

IMMDigitalVideo

playFast

Plays the video at the fast speed from the start position to the end position with no audio. If *from* is omitted, play starts at the current position; if *to* is omitted, play stops at the end of the video file.

```
IMMDigitalVideo&  
    playFast( const IMMTime& from = IMMTime ( ),  
              const IMMTime& to = IMMTime ( ),  
              CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the video file.

playScan

Plays frames when indexed; otherwise, it plays the video as fast as possible (without disabling the video) from the start position to the end position with no audio. If *from* is omitted, play starts at the current position; if *to* is omitted, play stops at the end of the video file.

```
IMMDigitalVideo&  
    playScan( const IMMTime& from = IMMTime ( ),  
              const IMMTime& to = IMMTime ( ),  
              CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the video file.

IMMDigitalVideo

playSlow

Plays the video at the slow speed from the start position to the end position with no audio. If *from* is omitted, play starts at the current position; if *to* is omitted, play stops at the end of the video file.

```
IMMDigitalVideo&
playSlow( const IMTime& from = IMTime ( ),           Win PM Motif
          const IMTime& to = IMTime ( ),              Y   Y   N
          CallType call = nowait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the video file.

Playback Speeds

Use these members to query the device's playback rates and the current playback rate.

fastSpeed

Returns the device's fast playback rate in the current speed format, either as a percentage or in frames-per-second. Returns 0 if the device cannot play fast.

```
IMMSpeed
fastSpeed( CallType call = wait ) const;           Win PM Motif
                                                    N   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

fileNormalSpeed

Returns the normal play rate of the currently loaded video file, in the current speed format, either as a percentage or in frames-per-second. This is the playback speed

IMMDigitalVideo

stored in the video file when the video file was created. Returns 0 if no file is currently loaded.

IMMSpeed	<u>Win</u>	<u>PM</u>	<u>Motif</u>
fileNormalSpeed(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

maximumSpeed

Returns the maximum play rate in the current speed format, either as percentage or in frames-per-second.

IMMSpeed	Win	PM	Motif
maximumSpeed(CallType call = wait) const;	<u>Y</u>	<u>Y</u>	<u>N</u>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

minimumSpeed

Returns the minimum play rate in the current speed format, either as percentage or in frames-per-second.

IMMSpeed	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minimumSpeed(CallType call = wait) const;	Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.

IMMDigitalVideo

Exceptions	
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

normalSpeed Returns the device's normal play rate in the current speed format, either as a percentage or in frames-per-second.

IMMSpeed
normalSpeed(CallType call = wait) const; Win PM Motif
Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

slowSpeed Returns the device's slow playback rate in the current speed format, either as a percentage or in frames-per-second. Returns 0 if the device cannot play slow.

IMMSpeed
slowSpeed(CallType call = wait) const; Win PM Motif
N Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

speed Returns the currently set speed for the device.

IMMSpeed
speed(CallType call = wait) const; Win PM Motif
Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMDigitalVideo

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Record and Monitoring

Use these members to record from a video capture card, monitor the video signal, and change the monitor window.

disableMonitoring

Sets monitoring of the incoming video signal off. When monitoring is turned on, a monitor window is created. When the monitor window is active and recording is not in progress, the monitor window displays the entire video source image, regardless of any source rectangle setting. During recording, only the area being captured is displayed.

Monitoring during real-time recording is supported but at a reduced performance. Monitoring cannot be turned on or off during recording, that is, if it is on when recording starts it must remain on while recording is in progress. Attempting to turn monitoring on or off during real-time recording results in an exception.

During monitoring, audio is passed through and heard on the speakers or headphones connected to the sound card, if present.

virtual IMMDigitalVideo&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
disableMonitoring(CallType call = wait);	<i>N</i>	<i>Y</i>	<i>N</i>

enableMonitoring

Sets monitoring of the incoming video signal on or off. When monitoring is turned on, a monitor window is created. When the monitor window is active and recording is not in progress, the monitor window displays the entire video source image, regardless of any source rectangle setting. During recording, only the area being captured is displayed.

Monitoring during real-time recording is supported but at a reduced performance. Monitoring cannot be turned on or off during recording, that is, if it is on when recording starts, it must remain on while recording is in progress. Attempting to turn monitoring on or off during real-time recording results in an exception.

IMMDigitalVideo

During monitoring, audio is passed through and heard on the speakers or headphones connected to the sound card, if present.

<code>virtual IMMDigitalVideo& enableMonitoring(Boolean enable = true, CallType call = wait);</code>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>Y</i>	<i>N</i>					

isMonitoringEnabled

Returns true if monitoring of the incoming video signal is enabled.

<code>Boolean isMonitoringEnabled(CallType call = wait) const;</code>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>Y</i>	<i>N</i>					

monitorHandle

Returns the window handle for the current video monitor window.

<code>IWindowHandle monitorHandle(CallType call = wait) const;</code>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>Y</i>	<i>N</i>					

record

Starts recording till it reaches the end location. If there was anything previously recorded, it is deleted before the recording starts. If the *end* location is not specified, recording continues until a pause or stop occurs. It is recommended that you temporarily acquire the device exclusively while you are recording; otherwise, the recording can become inactive if another device requests access to the device.

<code>virtual IMMDigitalVideo& record(const IMMTime& end = IMMTime (), Boolean resumeIfPaused = true, CallType call = nowait);</code>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setMonitorWindow

Supplies a window where you can see the incoming video signal if monitoring is enabled. For example, you might want your monitor window in a canvas. If so, pass the handle of the canvas to this function. By default, the IMMDigitalVideo class creates and uses its own video monitor window. This function allows an application to provide its own video monitor window. To switch back to using the default video monitor window, call `useDefaultMonitorWindow` (p. 568).

1 <code>IMMDigitalVideo& setMonitorWindow(const IWindowHandle& handle, CallType call = wait);</code>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>Y</i>	<i>N</i>					

IMMDigitalVideo

2	<code>IMMDigitalVideo& setMonitorWindow(const IWindow& window, CallType call = wait);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>N</i>	<i>Y</i>	<i>N</i>						

useDefaultMonitorWindow

Specifies that the digital video class creates and manages its own video monitor window. You can use it to set the video monitor window back to the default video monitor window.

<code>IMMDigitalVideo& useDefaultMonitorWindow(CallType call = wait);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>N</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>Y</i>	<i>N</i>					

Video Window

These members are used to change the video window attributes and to play the loaded video at a different rate than normal.

destinationRectangle

Returns the subrectangle within the video window where video is played. This allows an application to move, grow, or shrink where the video is played inside the video window.

<code>IRectangle destinationRectangle(CallType call = wait) const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

handle

Returns the window handle for the current video playback window.

<code>IWindowHandle handle(CallType call = wait) const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMDigitalVideo

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

maximumWindows

Returns the maximum number of video windows allowed on this machine.

```
unsigned long
maximumWindows( CallType call = wait ) const;           Win PM Motif
                                                         Y   Y   N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

refresh

Restores the video window by causing the video device to transfer an image from the video to the playback window. This is similar to a refresh of a normal window. If the source rectangle is not specified, the entire image is restored. If the destination rectangle is not specified, the destination size is set to the same as the image size. If the destination rectangle is a different size than the source rectangle, it tries to scale the image (if supported); otherwise, it clips the image to the destination rectangle.

```
IMMDigitalVideo&
refresh( const IRectangle& source = IRectangle ( ),      Win PM Motif
         const IRectangle& destination = IRectangle ( ),  N   Y   N
         CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

IMMDigitalVideo

Exceptions	
IAccessError	Invalid rectangle was specified.

setDestination

Sets the subrectangle within the video window where video is played. This allows an application to move, grow, or shrink where the video is played inside the video window.

```
IMMDigitalVideo&  
    setDestination( const IRectangle& rectangle,  
                   CallType = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid rectangle was specified.

setWindow

Supplies a window where the video is played. For example, you might want your video to be played back in a canvas. If so, pass the handle of the canvas to this function. By default, the IMMDigitalVideo class creates and use its own video window. This function allows an application to provide its own video playback window. To switch back to using the default video window, call useDefaultWindow (p. 571).

i

```
IMMDigitalVideo&  
    setWindow( const IWindowHandle& handle,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid window handle was specified.

IMMDigitalVideo

2 IMMDigitalVideo&
 setWindow(const IWindow& window,
 CallType call = wait);
Win **PM** **Motif**
 Y Y N

Win The video window will not resize to fit the size of the window that you specified. Please use the setDestination() member function to accomplish this.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid window was specified.

sourceRectangle

Returns the subrectangle of the video from the currently loaded video file.

IRectangle
 sourceRectangle(CallType call = wait) const;
Win **PM** **Motif**
 Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

useDefaultWindow

Specifies that the digital video class creates and manages its own video window. You can use it to set the video window back to the default video window.

IMMDigitalVideo&
 useDefaultWindow(CallType call = wait);
Win **PM** **Motif**
 Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.

IMMDigitalVideo

Exceptions	
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

videoFileHeight

Returns the height of the video for the currently loaded video file.

```
unsigned long  
videoFileHeight( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

videoFileName

Returns the file name of the digital video file.

```
IString  
videoFileName() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

videoFileWidth

Returns the width of the video for the currently loaded video file.

```
unsigned long  
videoFileWidth( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMDigitalVideo

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

videoHeight Returns the nominal (reasonable) height of the video for this machine. So, this would probably return a smaller value on a '386 machine than a '486 machine.

```
unsigned long  
videoHeight( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

videoWidth Returns the nominal (reasonable) width of the video for this machine. So, this would probably return a smaller value on a '386 machine than a '486 machine.

```
unsigned long  
videoWidth( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Inherited Public Functions

IMMConfigurableAudio		
bitsPerSample	format	setBytesPerSecond

IMMDigitalVideo

IMMConfigurableAudio		
blockAlignment	samplesPerSecond	setChannels
bytesPerSecond	setBitsPerSample	setFormat
channels	setBlockAlignment	setSamplesPerSecond

IMMRecordable		
canRedo	cut	redo
canUndo	deleteSelection	save
copy	paste	saveAs
cueForRecording	record	undo

IMMFileMedia		
filename	load	open
isWriteable	loadOnThread	openOnThread

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming

IMMDigitalVideo

IMMDevice		
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class.

IMMDigitalVideo

```
IMMDigitalVideo( unsigned long deviceIdentifier,           Win PM Motif
                  const IString& newAlias );               Y  Y  N
```

A derived class can construct an IMMDigitalVideo from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

IMMDigitalVideo

Implementation

These members validate the open string, translate the audio flags, and play the video file using the passed-in speed and range.

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

```
virtual Boolean  
    isOpenStringValid( const IString& deviceName ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

playDigital

Allows the playing from a start to an end position with the specified flags and speed. If *from* is omitted, play starts at the current position; if *to* is omitted, play stops at the end of the video file.

```
IMMDigitalVideo&  
    playDigital( const IMMTime& from = IMMTime ( ),  
                const IMMTime& to = IMMTime ( ),  
                CallType call = nowait,  
                unsigned long flags = 0,  
                unsigned long speed = 0 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the video file.

translateAudioFlag

Maps the wave audio commands to the digital video values. The digital video and the wave audio device both support setting the bytes-per-second, but the actual system values are different. By allowing the mapping of the wave audio values to the digital video values, the digital video player does not have to override all of the audio functions because they call this function.

```
virtual unsigned long  
    translateAudioFlag( unsigned long ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMConfigurableAudio		
translateAudioFlag		

IMMFileMedia		
enableDataUpdate	open	openOnThread

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IMMDigitalVideo

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMErrorInfo

Derivation

```

IBase
IVBase
IBaseErrorInfo
IMMErrorInfo
  
```

Inherited By None.

Header File immexcpt.hpp

Members	Member	Page	Member	Page
	Constructor	580	text	581
	errorId	581	throwMMError	581
	isAvailable	581	~IMMErrorInfo	581
	operator const char *	581		

The IMMErrorInfo class represents error information that you can include in an exception object. When an multimedia call results in an error condition, objects of the IMMErrorInfo class are created. You can use the error text to construct a derived class object of IException (Vol. I).

The User Interface Class Library provides the following macros for throwing exceptions constructed with IMMErrorInfo information.

ITHROWMMERROR

Accepts the following parameters:

error The multimedia error number.

functionName

The name of the multimedia function that returned an error condition.

This macro then generates code that calls IMMErrorInfo::throwMMError (p. 581), which does the following:

1. Creates an IMMErrorInfo object
2. Uses the object to create an object of IAccessError (Vol. I)
3. Adds location information
4. Logs the exception data
5. Throws the exception

IMMErrorInfo

Note: This macro uses the recoverable enumerator provided by `IException::Severity` (Vol. I).

ITHROWMMERROR2

Throws any of the User Interface Class Library-defined exceptions. This macro accepts the following parameters:

<i>error</i>	The multimedia error number.
<i>location</i>	The name of the multimedia function returning an error code, the name of the file the function is in, and the function's line number.
<i>type</i>	Use the enumeration <code>IBaseErrorInfo::ExceptionType</code> (Vol. I) to specify the type of the exception. The User Interface Class Library uses <code>accessError</code> as the default type on its exceptions.
<i>severity</i>	Use the enumeration <code>IException::Severity</code> (Vol. I) to specify the severity of the error. The User Interface Class Library uses <code>recoverable</code> as the default severity on its exceptions.

This macro generates code that calls `throwMMError` (p. 581), which does the following:

1. Creates an `IMMErrorInfo` object
2. Uses the object to create an `IException` object
3. Adds location information
4. Logs the exception data
5. Throws the exception



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot create copies of objects of this class.

IMMErrorInfo There is only one way to construct instances of this class. If the error number and the name of the failing API is passed in, it will be prefixed to the error text. If the error text cannot be loaded, the following default text is provided: "No error text is available."

```
IMMErrorInfo( unsigned long errorId = 0,  
              const char* functionName = 0 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IMMErrorInfo

~IMMErrorInfo

```
virtual  
~IMMErrorInfo();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Error Information

Use these members to query the accessible attributes of this class.

errorId Returns the error ID.

```
virtual unsigned long  
errorId() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isAvailable If there is error text available for the error, returns true.

```
virtual Boolean  
isAvailable() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

operator const char *

Returns the error text.

```
virtual  
operator const char *() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

text Returns the error text.

```
virtual const char*  
text() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Throw Support

Use these members to support throwing of exceptions. The ITHROWMMERROR macro uses these members.

throwMMError

Creates an IMMErrorInfo object and uses the text from it to do the following:

1. Create an exception object
2. Add the location information to it
3. Log the exception data
4. Throw the exception

IMMErrorInfo

- errorId* The error number for the exception.
- functionName* The name of the function (probably a system function) that failed.
- location* An IExceptionLocation (Vol. I) object containing the following:
- Function name where the failure was detected and the exception thrown
 - File name
 - Line number where the function is called
- name* Use the enumeration IBaseErrorInfo::ExceptionType (Vol. I) to specify the type of the exception. The default is accessError.
- severity* Use the enumeration IException::Severity (Vol. I) to specify the severity of the error. The default is recoverable.

```
static void                                     Win PM Motif
    throwMMErrors(                             Y  Y  N
        unsigned long errorId,
        const char* functionName,
        const IExceptionLocation& location,
        IBaseErrorInfo::ExceptionType name = accessError,
        IException::Severity severity = recoverable );
```

Inherited Public Functions

IBaseErrorInfo		
errorId	isAvailable	operator const char *

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMFileMedia

IMMFileMedia

Derivation

IBase
IVBase
INotifier
IStandardNotifier
IMMDevice
IMMPlayableDevice
IMMFileMedia

Inherited By IMMRecordable
IMMSequencer

Header File immfilem.hpp

Members	Member	Page	Member	Page
	Constructor	589	loadOnThread	586
	enableDataUpdate	590	open	587
	filename	585	openOnThread	587
	isWriteable	585	~IMMFileMedia	584
	load	585		

The IMMFileMedia is the base class for devices that use files, and it provides functions for loading data from a file.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

Derived classes can use these members to create objects of this class.

~IMMFileMedia

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IMMFileMedia();	Y	Y	N

File Loading

Use these members to load a file into the device and to query information about the currently loaded file, for example, the file name and if the file was loaded as read-only.

filename Returns the currently loaded file name.

```

IString
    filename( CallType call = wait ) const;

```

Win PM Motif
 Y Y N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file was not found; possibly there is no loaded file.

isWriteable Returns false if the file was loaded as read-only; otherwise, true is returned. If the file was loaded as read-only, then the system prevents any changes to the file. The audio driver might also be able to improve load and runtime performance as no changes are allowed. Save and record functions throw an exception if the file was loaded as read-only.

```

virtual Boolean
    isWriteable() const;

```

Win PM Motif
 Y Y N

load Loads the given file into memory. If the file does not exist, a temporary file is created. Note, if *call* is wait (p. 543), then the windowing system is tied up until the file is loaded. If you do not want to tie up of the windowing system and you do not want the load function to return until the file is loaded, use the loadOnThread (Vol. II) function. Otherwise, you could set *call* to nowait (p. 543), which causes this function to return immediately. When the load finishes, an INotificationEvent (Vol. I) is sent to any observers that are attached to this device, notifying them that the load has finished.

If the file is loaded as read-only, the system prevents any changes being made to the file. The audio driver might also be able to improve load and runtime performance as no changes are allowed. Save and record functions throw an exception if the file was loaded as read-only.

IMMFileMedia

```
virtual IMMFileMedia&
load( const IString& filename,
      Boolean readOnly = false,
      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found.
IAccessError	Invalid media type or invalid data format.
IAccessError	A hardware error occurred.
IAccessError	A file attribute error occurred.
IAccessError	This format is unsupported; possibly the hardware does not support the samples-per-second, the bits-per-sample, the number of channels, or the audio format.

loadOnThread

Loads the given file into memory by creating a thread to do the actual loading. By doing the loading this way, it does not tie up the windowing system. This function does not return until the file is loaded. If the file does not exist, a temporary file is created.

If the file is loaded as read-only, the system prevents any changes being made to the file. The audio driver might also be able to improve load and runtime performance as no changes are allowed. Save and record functions throw an exception if the file was loaded as read-only.

```
virtual IMMFileMedia&
loadOnThread( const IString& filename,
              Boolean readOnly = false );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The Win32 MCI does not support threads, so this function behaves like load() in this environment.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.

IMMFileMedia

Exceptions	
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found.
IAccessError	Invalid media type or invalid data format.
IAccessError	A hardware error occurred.
IAccessError	A file attribute error occurred.
IAccessError	This format is unsupported; possibly the hardware does not support the samples-per-second, the bits-per-sample, the number of channels, or the audio format.

open Opens the device.

```
virtual IMMFileMedia&
    open( const IString& fileOrDevice = IString ( ),           Win PM Motif
          Boolean shareable = true,                             Y  N  N
          CallType call = wait );
```



This member is overridden in this derived class for specific operating system behavior.

openOnThread

Opens the device.

```
virtual IMMFileMedia&
    openOnThread( const IString& fileOrDevice = IString ( ),   Win PM Motif
                  Boolean shareable = true,                     Y  N  N
                  CallType call = wait );
```



This member is overridden in this derived class for specific operating system behavior.

Inherited Public Functions

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMFileMedia

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

Derived classes can use these members to create objects of this class.

IMMFileMedia

IMMFileMedia

1 IMMFileMedia(const IString& deviceName,
Boolean openNow,
unsigned long instance,
Boolean openShareable);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMFileMedia from the following:

deviceName

You must specify what device you want to construct.

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.

instance You can provide your own instance number instead of letting IMMFileMedia generate one.

openShareable

If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

2 IMMFileMedia(unsigned long deviceIdentifier,
const IString& newAlias = IString ());

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMFileMedia from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

File Loading

Use these members to load a file into the device and to query information about the currently loaded file, for example, the file name and if the file was loaded as read-only.

open Opens the device.

virtual IMMFileMedia&
open(unsigned long instanceNumber,
Boolean shareable = true,
CallType call = nowait);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



This member is overridden in this derived class for specific operating system behavior.

IMMFileMedia

Opens the device.

openOnThread

```
virtual IMMFileMedia&
  openOnThread( unsigned long instanceNumber,
                Boolean shareable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>



This member is overridden in this derived class for specific operating system behavior.

Implementation

These members change the read-only state of the flag for the loaded file.

enableDataUpdate

Allows the device to track read-only status of the loaded file.

```
virtual IMMFileMedia&
  enableDataUpdate( Boolean update = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMHourMinSecFrameTime

IMMHourMinSecFrameTime

Derivation IBase
IVBase
IMMTime
IMMHourMinSecFrameTime

Inherited By IMM24FramesPerSecondTime
IMM25FramesPerSecondTime
IMM30FramesPerSecondTime

Header File immtime.hpp

Members	Member	Page	Member	Page
	Constructor	593	operator unsigned long	594
	asString	594	ordinal	595
	frames	592	seconds	593
	framesPerSecond	592	setTimeToOrdinal	595
	hours	593	~IMMHourMinSecFrameTime	594
	minutes	593		

The IMMHourMinSecFrameTime data type class represents the hours-minutes-seconds-frames (HHMMSSFF) time format.

W32s The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Attributes

Use these functions to parse the time into normal time values, for example, hours, minutes, and seconds.

frames Returns the frames component of the time.

virtual unsigned long frames() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

framesPerSecond
Returns the number of frames per second.

IMMHourMinSecFrameTime

	unsigned long framesPerSecond() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
hours	Returns the hours component of the time.			
	virtual unsigned long hours() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
minutes	Returns the minutes component of the time. This is in the range of 0 to 59.			
	virtual unsigned long minutes() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
seconds	Returns the seconds component of the time. This is in the range of 0 to 59.			
	virtual unsigned long seconds() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

Constructors

You can construct and destruct objects of this class.

IMMHourMinSecFrameTime

1	IMMHourMinSecFrameTime(const IMMHourMinSecFrameTime& time);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
2	IMMHourMinSecFrameTime(unsigned long value = defaultTime, unsigned long framesPerSecond = 1);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

You can construct an IMMHourMinSecFrameTime from the following:

value A time value(FFSSMMHH) where:

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

framesPerSecond

The number of frames that occur for each second.

IMMHourMinSecFrameTime

~IMMHourMinSecFrameTime

```
virtual  
~IMMHourMinSecFrameTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long

Returns the time as an unsigned long in the following format (FFSSMMHH):

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the hours

```
virtual  
operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Diagnostics

Use these members to return the time as a string.

asString Returns the time value as a string formatted as HH:MM:SS.FF.

```
virtual IString  
asString() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Ordinal

Use these ordinal number functions for indicating generic positions. A time object returns an ordinal number, which represents the time at some particular granularity. This is useful for something that allows seeking and displays position, such as a slider. Because sliders generally do not process time values, you use an ordinal number to establish the range of slider values. Ordinals numbers can be mathematically manipulated and converted back to time values, if necessary.

Note: Some time values are difficult to represent as an ordinal value, such as IMMTrackMinSecFrameTime. Such a time object produces ordinal numbers that are dependent on unknown information, such as the table of contents of a CD. Some time classes might not be able to correctly implement setTimeToOrdinal (p. 595).

IMMHourMinSecFrameTime

ordinal Returns an ordinal number in frames. This is the time converted to the total number of frames based on the number of frames per second.

```
virtual unsigned long  
ordinal() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setTimeToOrdinal Sets the time object to the value represented by the ordinal number (in frames). This uses the current number of frames-per-second to convert this value.

```
virtual IMMTime&  
setTimeToOrdinal( unsigned long ordinal );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

IMMHourMinSecFrameTime

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMHourMinSecTime

Derivation

- IBase
- IVBase
- IMMTime
- IMMHourMinSecTime

Inherited By None.

Header File immtime.hpp

Members	Member	Page	Member	Page
	Constructor	598	ordinal	599
	asString	599	seconds	597
	hours	597	setTimeToOrdinal	599
	minutes	597	~IMMHourMinSecTime	598
	operator unsigned long	598		

The IMMHourMinSecTime datatype class represents the hours-minutes-seconds (HHMMSS) time format.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Attributes

Use these functions to parse the time into normal time values, for example, hours, minutes, and seconds.

hours Returns the hours component of the time.

virtual unsigned long	Win	PM	Motif
hours() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

minutes Returns the minutes component of the time. This is in the range of 0 to 59.

virtual unsigned long	Win	PM	Motif
minutes() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

seconds Returns the seconds component of the time. This is in the range of 0 to 59.

IMMHourMinSecTime

```
virtual unsigned long
seconds() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

IMMHourMinSecTime

```
1 IMMHourMinSecTime( unsigned long value = defaultTime );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMHourMinSecTime from the following:

- value* A time value where:
- 1. 1st byte is reserved(and not used)
 - 2. 2nd byte is the seconds
 - 3. 3rd byte is the minutes
 - 4. 4th byte is the hours

```
2 IMMHourMinSecTime( const IMMHourMinSecTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

```
3 IMMHourMinSecTime( const IMMTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMHourMinSecTime

```
virtual
~IMMHourMinSecTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long

Returns the time as a unsigned long in the following format (RRSSMMHH):

- 1. 1st byte is reserved
- 2. 2nd byte is the seconds
- 3. 3rd byte is the minutes

IMMHourMinSecTime

4. 4th byte is the hours

```
virtual  
operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Diagnostics

Use these members to return the time as a string.

asString Returns the time value as a string formatted as HH:MM:SS.

```
virtual IString  
asString() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Ordinal

Use these ordinal number functions for indicating generic positions. A time object returns an ordinal number, which represents the time at some particular granularity. This is useful for something that allows seeking and displays position, such as a slider. Because sliders generally do not process time values, use an ordinal number to establish the range of slider values. Ordinal numbers can be mathematically manipulated and converted back to time values, if necessary.

Note: Some time values are difficult to represent as an ordinal value, such as IMMTrackMinSecFrameTime. Such a time object produces ordinal numbers that are dependent on unknown information, such as the table of contents of a CD. Some time classes might not be able to correctly implement setTimeToOrdinal (p. 599).

ordinal Returns the time value in total seconds.

```
virtual unsigned long  
ordinal() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setTimeToOrdinal

Sets the time object to the value represented by the ordinal number (in seconds).

```
virtual IMMTime&  
setTimeToOrdinal( unsigned long ordinal );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMHourMinSecTime

Inherited Public Functions

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMMasterAudio

Derivation IBase
IVBase
INotifier
IStandardNotifier
IMMMasterAudio

Inherited By None.

Header File immmaud.hpp

Members	Member	Page	Member	Page
	Constructor	601	saveHeadphonesSetting	602
	areHeadphonesEnabled	602	saveSpeakersSetting	603
	areSpeakersEnabled	603	saveVolume	603
	disableHeadphones	602	setVolume	603
	disableSpeakers	603	volume	604
	enableHeadphones	602	~IMMMasterAudio	602
	enableSpeakers	603		

The IMMMasterAudio class provides behavior for modifying the master audio settings for all audio devices in the system. The application can set, query and save the headphones', speakers', and master volume settings for the system. When any of the multimedia devices are opened, they query the master audio settings and automatically adjust their settings accordingly.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMMasterAudio

IMMMasterAudio();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IMMMasterAudio

~IMMMasterAudio

```
virtual  
~IMMMasterAudio();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Headphones

Use these members to set, query, and save the headphones' setting.

areHeadphonesEnabled

Returns if the headphones' setting is enabled for the passed-in source (either the current setting or the saved setting); otherwise, returns false.

```
Boolean  
areHeadphonesEnabled(  
    SettingSource source = current,  
    IMMDevice::CallType call = IMMDevice::wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

disableHeadphones

Turns the headphones' settings off.

```
virtual IMMMasterAudio&  
disableHeadphones(  
    IMMDevice::CallType call = IMMDevice::wait);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

enableHeadphones

Turns the headphones' settings either off or on.

```
virtual IMMMasterAudio&  
enableHeadphones(  
    Boolean enable = true,  
    IMMDevice::CallType call = IMMDevice::wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

saveHeadphonesSetting

Saves the current headphones' setting of the operating system.

```
virtual IMMMasterAudio&  
saveHeadphonesSetting(  
    IMMDevice::CallType call = IMMDevice::wait);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Speakers

Use these members to set, query, and save the speakers' setting.

areSpeakersEnabled

Returns true if the speakers' setting is enabled for the passed-in source (either the current setting or the saved setting); otherwise, returns false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
areSpeakersEnabled(SettingSource source = current, IMMDevice::CallType call = IMMDevice::wait) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

disableSpeakers

Turns the speakers' setting off.

virtual IMMMasterAudio& disableSpeakers(IMMDevice::CallType call = IMMDevice::wait);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

enableSpeakers

Turns the speakers' setting either on or off.

virtual IMMMasterAudio& enableSpeakers(Boolean enable = true, IMMDevice::CallType call = IMMDevice::wait);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

saveSpeakersSetting

Saves the current speakers' setting of the operating system.

virtual IMMMasterAudio& saveSpeakersSetting(IMMDevice::CallType call = IMMDevice::wait);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Volume

Use these members to set, query, and save the master volume setting.

saveVolume Saves the current master volume setting of the operating system.

virtual IMMMasterAudio& saveVolume(IMMDevice::CallType call = IMMDevice::wait);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setVolume Sets the master volume for the operating system.

IMMMasterAudio

```
virtual IMMMasterAudio&
  setVolume( unsigned long volume,
             IMMDevice::CallType call = IMMDevice::wait );
```

Win

PM

Motif

Y

Y

N

volume Returns the master volume setting for either the current setting or the saved setting to a percent of the maximum audio level.

```
unsigned long
  volume( SettingSource source = current,
          IMMDevice::CallType call = IMMDevice::wait ) const;
```

Win

PM

Motif

Y

Y

N

Inherited Public Functions

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IStandardNotifier		
deleteId		

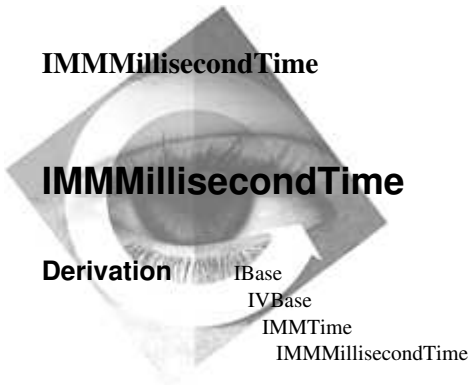
Inherited Protected Data

IBase		
recoverable	unrecoverable	

SettingSource

```
SettingSource {
    saved = 0x00004000L,
    current = 0x00008000L
};
```

Enumeration that specifies which setting to query and set.




Inherited By None.

Header File immtime.hpp

Members	Member	Page
	Constructor	606
	operator unsigned long	607
	~IMMMillisecondTime	607

The IMMMillisecondTime data type class represents one-thousandth of a second.

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMMillisecondTime

1	IMMMillisecondTime(unsigned long time = defaultTime);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>
2	IMMMillisecondTime(const IMMMillisecondTime& time);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>
3	IMMMillisecondTime(const IMTime& time);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

IMMMillisecondTime

~IMMMillisecondTime

```
virtual  
~IMMMillisecondTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long

Returns the time as an unsigned long where each time unit is equal to one-thousandth of a second.

```
virtual  
operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IMMMillisecondsTime

Inherited Protected Functions

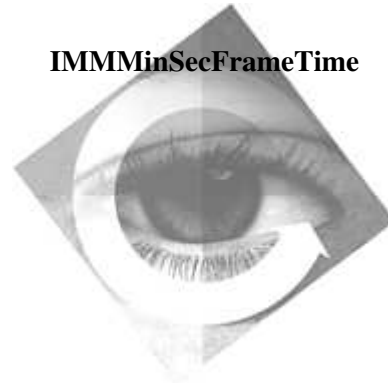
IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMMinSecFrameTime

Derivation

```

IBase
IVBase
IMMTime
IMMMinSecFrameTime
  
```

Inherited By None.

Header File immtime.hpp

Members	Member	Page	Member	Page
	Constructor	610	operator unsigned long	610
	asString	611	seconds	609
	frames	609	~IMMMinSecFrameTime	610
	minutes	609		

The IMMMinSecFrameTime data type class represents the time format based on the 75-frames-per-second CD digital audio standard.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Attributes

Use these members to parse the time into normal time values, for example, hours and minutes.

frames Returns the frames component of the time.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
frames() const;	Y	Y	N

minutes Returns the minutes component of the time. This is in the range of 0 to 59.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minutes() const;	Y	Y	N

seconds Returns the seconds component of the time. This is in the range of 0 to 59.

IMMMinSecFrameTime

```
unsigned long  
seconds() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

IMMMinSecFrameTime

```
1 IMMMinSecFrameTime( const IMMMinSecFrameTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

```
2 IMMMinSecFrameTime( unsigned long value = defaultTime );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMMinSecFrameTime from the following:

value A time value where:

1. 1st byte is the reserved (not used)
2. 2nd byte is the frames
3. 3rd byte is the seconds
4. 4th byte is the minutes

```
3 IMMMinSecFrameTime( const IMMTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMMinSecFrameTime

```
virtual  
~IMMMinSecFrameTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long.

operator unsigned long

Returns the time as a unsigned long in the following format (RRFFSSMM):

1. 1st byte is reserved
2. 2nd byte is the frames
3. 3rd byte is the seconds

IMMMinSecFrameTime

4. 4th byte is the minutes

```
virtual  
    operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Diagnostics

Use these members to return the time as a string.

asString Returns the time value as a string formatted as MM:SS.FF.

```
virtual IString  
    asString() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

IMMMinSecFrameTime

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMNotifyEvent

Derivation

```

IBase
IVBase
IEvent
IMMNotifyEvent
  
```

Inherited By None.

Header File immevt.hpp

Members					
	Member	Page		Member	Page
	Constructor	614		errorText	614
	command	613		successCode	615
	device	614		userParameter	614
	errorId	614		~IMMNotifyEvent	614

The IMMNotifyEvent class is the class for notification events. These events get generated when any device function gets called. This can be powerful for asynchronous processing when you do not want to wait for something to finish. For more information, see IMMDevice::nowait (p. 543). To get notified of events, you attach an observer to the device, and it sends a notification when a command has finished processing. From this event, you can find out what command was called, from which device it came, the return value of the command that was executed, and, if the command generated an error, what its error number and string are.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Command

Use these members to determine what command generated this notification.

command Returns the command that generated this notification event.

```

Command
command() const;
  
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMNotifyEvent

Constructors

You can construct and destruct objects of this class.

IMMNotifyEvent

Although you can construct objects of this class, typically IMMDeviceHandler::dispatchHandlerEvent (p. 552) creates objects of this class from an object of the class IEvent (Vol. II) or another IMMNotifyEvent object.

IMMNotifyEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMNotifyEvent

virtual ~IMMNotifyEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to parse the event information into the success state of the command (and, if there is an error, its error code and string) and to access the user parameter.

device Returns a pointer to the device that this event is from.

IMMDevice* device() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

errorId Returns the error code from the command if there was one; otherwise, it returns 0.

unsigned short errorId() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

errorText Returns the error text for a given error code. If there was not any error text, then it returns a null IString.

static IString errorText(unsigned long errorCode);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

userParameter

Returns the user parameter that was set at the time of this event. See IMMDevice::setUserParameter (p. 537) for more information.

IMMNotifyEvent

```
unsigned short  
userParameter() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Success Code

Use these members to determine if the command succeeded.

successCode Returns the success code, which indicates if the command succeeded or failed.

```
SuccessCode  
successCode() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMMNotifyEvent

```
Command {
    open = 1,          close,          escape,
    play,              seek,           stop,
    pause,             info,          getDeviceCapabilities,
    status,            spin,         set,
    step,              record,        sysInfo,
    save,              cue,          update,
    setCuePoint,       setPositionAdvise, setSyncOffset,
    load,              acquireDevice,  releaseDevice,
    masterAudio,       getTableOfContents, deviceSettings,
    connector,         resume,        connectorInfo,
    defaultConnection, connection,    group,
    capture = 40,      freeze,       getImageBuffer,
    getImagePalette,   put,          realize,
    rewind,            restore,      setImageBuffer,
    setImagePalette,   unfreeze,     where,
    windowCommand,     deleteCommand, cut,
    paste,             copy,         redo,
    undo,              breakCommand, monitor,
    reserve,           setAudioCommand, signal,
    setVideoCommand,   qualityCommand, listCommand,
    configureCommand
};
```

Enumeration that specifies the command that this event was generated for.

SuccessCode

```
SuccessCode {
    successful,
    preempted,
    aborted,
    error
};
```

Enumeration that specifies if the command was successful. The following are valid values:

successful

The command was completed successfully.

preempted

Another notification request (same type of command) was received.

aborted

The command was interrupted and is unable to be completed. For example, the first command was a play with nowait, and a second command, stop, was issued before the play could finish.

error

The command caused an error to occur.



IMMPassDeviceEvent


Derivation IBase
 IVBase
 IEvent
 IMMPassDeviceEvent

Inherited By None.

Header File immevt.hpp

Members	Member	Page	Member	Page
	Constructor device	617 618	isGainingUse ~IMMPassDeviceEvent	618 617

The IMMPassDeviceEvent class is the class for pass device events. These events get generated when a device is being requested to release a shared hardware device (such as a CD-ROM drive).

 The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMPassDeviceEvent

Although you can construct objects of this class, typically
IMMDeviceHandler::dispatchHandlerEvent (p. 552) creates objects of this class from
an object of the class IEvent (Vol. II) or another IMMPassDeviceEvent object.

IMMPassDeviceEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMPassDeviceEvent

virtual ~IMMPassDeviceEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IMMPassDeviceEvent

Event Information

Use these members to parse the event into the ID of the device that is either losing or gaining use of the hardware device.

device Returns a pointer to the device that is being requested to lose use of or gain use of a device. See isGainingUse (p. 618) to find out what this device is being requested to do.

IMMDevice*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
device() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isGainingUse Returns true if the device is gaining use of the hardware device and returns false if it is losing use of the device.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isGainingUse() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMPlayableDevice

IMMPlayableDevice

Derivation

IBase
IVBase
INotifier
IStandardNotifier
IMMDevice
IMMPlayableDevice

Inherited By

IMMFileMedia
IMMRemovableMedia

Header File

immplayd.hpp

Members

Member	Page	Member	Page
Constructor	628	seek	626
addCuePoint	620	seekToEnd	626
cueForPlayback	624	seekToStart	626
length	621	startPositionTracking	622
pause	623	stepFrame	627
play	624	stop	625
position	622	stopPositionTracking	623
removeCuePoint	621	~IMMPlayableDevice	625
resume	623		

The IMMPlayableDevice is the base class for playable devices. It provides all of the common base functions for play, pause, stop, and similar functions. There are functions for querying and moving the current position and for turning on notification to observers of position changes.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Cue Points

Use these members to add and remove cue points.

addCuePoint Sets a cue point at the given location. A *cue point* is a location or time position in a media device. When a device encounters (by playing or recording) a time position associated with a cue point, a IMM_CuePointEvent (p. 509) is generated.

IMMPlayableDevice

```
virtual IMMPlayableDevice&
    addCuePoint( const IMMTime& time,
                 CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The <i>time</i> is an invalid time value.

removeCuePoint

Removes the given cue point. A *cue* point is a location or time position in a media device. When a device encounters (by playing or recording) a time position associated with a cue point, a IMM CuePointEvent (p. 509) is generated.

```
virtual IMMPlayableDevice&
    removeCuePoint( const IMMTime& time,
                   CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The <i>time</i> is an invalid value; possibly the time value does not have a cue point set at its location.

Notifications

Use start and stop position tracking members to cause position change events to be generated. Use the length function to determine the amount of data and the position function to find out where you are in the data.

length

Returns the total length in the current time format as an unsigned long. Use this value to create a time object in the current time format.

```
unsigned long
    length( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMPlayableDevice

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

position

Returns the current position in the current time format as an unsigned long. Use this value to create a time object in the current time format.

```
unsigned long
position( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

startPositionTracking

Starts position tracking. *Position tracking* causes a position change event to be generated every time the position of the media changes the passed in *timeInterval*.

```
virtual IMMPlayableDevice&
startPositionTracking( const IMTime& timeInterval,
                      CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
N	Y	N



Once you start position tracking for a device, you will continue to receive `positionChangeEvents` for that device even if it is stopped. To end position tracking, use `stopPositionTracking`.

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

IMMPlayableDevice

Exceptions	
IAccessError	The file is not found; possibly no file is currently loaded.
IAccessError	The <i>timeInterval</i> is out of range; possibly the time value is past the length of the data.

stopPositionTracking

Stops position tracking. *Position tracking* causes a position change event to be generated every time the position of the media changes a given time amount.

```
virtual IMMPlayableDevice&
    stopPositionTracking( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
N	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Pause and Resume

Use these members to pause and resume playback.

pause If the device is playing, then it pauses the device.

```
virtual IMMPlayableDevice&
    pause( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

resume Resumes playback of the device from a paused state if *resume* is true. The previous specified *to* parameter in the *play* (p. 624) function remains in effect.

IMMPlayableDevice

```
virtual IMMPlayableDevice&  
    resume( Boolean resume = true,  
            CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

Play and Stop

Use these members to play or stop playback of the device. Use `cueForPlayback` to notify the device that you are going to start to play the device.

cueForPlayback

Cues the device for playback. This function does not have to be issued before playback; however, depending on the device, it might reduce the delay associated with the play function. For example, you might have a compact disc player that stops the spinning of the CD, when it isn't playing. This function might cause the CD to start spinning, thereby reducing the time before the CD starts the playback.

```
virtual IMMPlayableDevice&  
    cueForPlayback( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IInvalidRequest	There is no data present; possibly no file is loaded or no media is in the device.

play

Starts playing the device from the passed-in start position to the passed-in end position. If *from* is omitted, the device starts playing at the current position; if *to* is omitted, play stops at the end of the data. If *resumeIfPaused* is true and the device is paused, it resumes playback of the device; otherwise, it starts playback from the passed in start position.

IMMPlayableDevice

```
virtual IMMPlayableDevice&
    play( const IMMTime& from = IMMTime ( ),
          const IMMTime& to = IMMTime ( ),
          Boolean resumeIfPaused = true,
          CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
InvalidRequest	There is no data present; possibly no file is loaded or no media is in the device.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the data.

stop Stops playback of the device.

```
virtual IMMPlayableDevice&
    stop( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Constructors

Derived classes can use these functions to create objects of the IMMPlayableDevice class.

~IMMPlayableDevice

```
virtual
    ~IMMPlayableDevice();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMPlayableDevice

Seek

Use these members to seek (move the current position) to either the beginning or the ending of the data or to a particular location in the data. Also, you can use the step frame member to move a single frame forward or backward.

seek Sets the current position of the device to the passed in position.

```
virtual IMMPlayableDevice&  
    seek( const IMMTime& to,  
          CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found; possibly no file is currently loaded.
IAccessError	The current device does not support the ability to seek.
IAccessError	The <i>to</i> position is out of range; possibly it is beyond the end of the media.

seekToEnd Moves the current position to the end of the data in the device.

```
virtual IMMPlayableDevice&  
    seekToEnd( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found; possibly no file is currently loaded.
IAccessError	The current device does not support the ability to seek.

seekToStart Moves the current position to the start of the data in the device.

IMMPlayableDevice

```
virtual IMMPlayableDevice&  
    seekToStart( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found; possibly no file is currently loaded.
IAccessError	The current device does not support the ability to seek.

stepFrame Steps the play one or more time units forward or backward.

```
virtual IMMPlayableDevice&  
    stepFrame( unsigned long frames = 1,  
              Boolean forward = true,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	This is an unsupported function; possibly the device does not support the ability to step the player.

Inherited Public Functions

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject

IMMPlayableDevice

IMMDevice		
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	


IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

Derived classes can use these functions to create objects of the IMMPlayableDevice class.

IMMPlayableDevice

 IMMPlayableDevice(const IString& deviceOrFileName,

Boolean openNow = true,

unsigned long instance = 0,

Boolean openShareable = true);

Win

PM

Motif

Y

Y

N

IMMPlayableDevice

A derived class can construct an IMMPlayableDevice from the following:

deviceOrFileName

You must specify what type of playable device you want to construct.

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.

instance You can provide your own instance number instead of letting IMMPlayableDevice generate one.

openShareable

If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

```
2 IMMPlayableDevice( unsigned long deviceIdentifier,      Win PM Motif
                      const IString& newAlias = IString ( ) );  Y   Y   N
```

A derived class can construct an IMMPlayableDevice from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Inherited Protected Functions

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

IMMPlayableDevice

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMMPlayerPanel

Derivation

- IBase
- IVBase
- INotifier
- IWindow
- IControl
- ICanvas
- IMultiCellCanvas
- IMMPlayerPanel

Inherited By None.

Header File immplypn.hpp

Members	Member	Page	Member	Page
	Constructor	633	rewindButton	633
	deviceType	635	setPlayableDevice	635
	fastForwardButton	632	stepBackwardButton	633
	pauseButton	633	stepForwardButton	633
	playableDevice	635	stopButton	633
	playButton	633	~IMMPlayerPanel	634

IMMPlayerPanel is a device player panel that contains various buttons for controlling the device. It can be constructed directly or inherited if finer device control or more functionality is needed. The user is presented with a generic interface, sufficient to control most multimedia devices. The following figure shows an IMMPlayerPanel with device type of digital video.



Figure 8. An IMMPlayerPanel with Device Type of Digital Video

Not only does it create the buttons and organize them, but it also makes the actual function call for the button. For example, the play button calls the play function on whatever device is currently set into the player panel. Also, there is some intelligent processing for disabling and enabling the buttons. For example, if the device is stopped, the stop and pause buttons are disabled. Or, if the device is playing and the user presses the pause button, the pause button stays pressed in the down (latched) state and the play button is unlatched. If the user then presses the pause button again, the pause button is unlatched and the play button is latched. The player panel

IMMPlayerPanel

automatically attaches an IMMPlayerPanelHandler (p. 638) object to provide default behavior. See IMMPlayerPanelHandler (p. 638) for more information on the default processing for each of the buttons and for how to change the default behavior.

If you create the IMMPlayerPanel without passing in a device type, you get the default buttons, which are rewind, stop, pause, play, and fast forward. If you pass in overlay, videoDisc animation, or digitalVideo, you also get step forward and step backward buttons.

IMMPlayerPanel is implemented using a set of buttons on an IMultiCellCanvas (Vol. III). The canvas is set up with all of the default behavior and styles of the multicell canvas, and none of the rows or columns is made expandable. The buttons are added to the multicell canvas in the following coordinates:

play	Cell 4,1 or 5,1 if step buttons are enabled
pause	Cell 3,1 or 4,1 if step buttons are enabled
fastForward	Cell 5,1 or 6,1 if step buttons are enabled
rewind	Cell 1,1 or 2,1 if step buttons are enabled
stop	Cell 2,1 or 3,1 if step buttons are enabled
stepForward	Cell 7,1 if step buttons are enabled
stepBackward	Cell 1,1 if step buttons are enabled

By knowing where the buttons are, you can easily extend this class by using addToCell (Vol. III) to add a vacant cell and to add a command handler to your object of this class to process the new buttons.

Note: The identifiers for the animated buttons are the same as the identifiers for the bitmaps, which are defined in *icconst.h*. They are IC_FF, IC_REWIND, IC_PAUSE, IC_STOP, IC_STEPF, IC_STEPB, and IC_PLAY1.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Buttons

Use these members to access the buttons on the player panel.

fastForwardButton

Returns a pointer to the fast forward animated button.

```
IAnimatedButton*  
    fastForwardButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IMMPlayerPanel

pauseButton Returns a pointer to the pause animated button.

```
IAnimatedButton*  
    pauseButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

playButton Returns a pointer to the play animated button.

```
IAnimatedButton*  
    playButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

rewindButton

Returns a pointer to the rewind animated button.

```
IAnimatedButton*  
    rewindButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

stepBackwardButton

Returns a pointer to the step backward animated button.

```
IAnimatedButton*  
    stepBackwardButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

stepForwardButton

Returns a pointer to the step forward animated button.

```
IAnimatedButton*  
    stepForwardButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

stopButton Returns a pointer to the stop animated button.

```
IAnimatedButton*  
    stopButton() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

IMMPlayerPanel

You can construct objects of this class using the following parameters:

IMMPlayerPanel

identifier Window identifier of the player panel you construct.

We recommend that you do one of the following:

- Give unique identifiers to all windows in the same frame window. “In the same window” means that the frame window is the first frame in their parent window chain.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID.

parent The parent window of the player panel you construct. You must specify a parent window. This constructor throws an `InvalidParameter` exception if you pass an `IWindow*` of 0. The parent window is primarily used for visible relationships.

owner The owner window of the player panel you construct.

deviceType The type of device the player panel is going to control. The main use of this is to determine the type of buttons to be displayed on the player panel. Optional.

initial The initial position and size of the player panel you construct. The initial position is the lower-left corner of the player panel relative to the lower-left corner of the parent window. Optional.

style Use the styles provided by `IMultiCellCanvas Style (Vol. III)` to specify the control’s styles. Optional.

```
IMMPlayerPanel(  
    unsigned long identifier,  
    IWindow* parent,  
    IWindow* owner,  
    unsigned long deviceType = IMMDevice::other,  
    const IRectangle& initial = IRectangle ( ),  
    const IMultiCellCanvas::Style& style =  
        IMultiCellCanvas::defaultStyle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>



The window identifier is limited to the range 0 through 65535.

The owner window is primarily used for routing notification events and unprocessed messages. The operating system also uses the owner window chain to inherit colors.

~IMMPlayerPanel

```
virtual  
~IMMPlayerPanel();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMPlayerPanel

Device

Use these members to query and set the device that the player panel is going to control and to query the type of device that is being controlled.

deviceType Returns the type of device this player panel was created for. See the valid device types in IMMDevice (p. 512).

```
unsigned long  
deviceType() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

playableDevice

Returns a pointer to the IMMPlayableDevice (p. 620) the player is controlling. This allows the application to determine which device is set into the player panel.

```
IMMPlayableDevice*  
playableDevice() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setPlayableDevice

Sets a pointer to the IMMPlayableDevice (p. 620) that the player panel will control. This allows the player panel to control almost any device that the user wants. For example, you might want to control a digital video device. If so, you would create an IMMDigitalVideo (p. 557) device and pass it on this call. When the user presses the player panel buttons, such as play or pause, these functions would act upon that IMMDigitalVideo device.

```
IMMPlayerPanel&  
setPlayableDevice( IMMPlayableDevice* device );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMultiCellCanvas		
addToCell	enableDragLines	rowHeight
columnWidth	enableGridLines	setColumnWidth
convertToGUIStyle	hasDragLines	setDefaultCell
defaultCell	hasGridLines	setDefaultStyle
defaultStyle	isColumnExpandable	setLayoutDistorted
disableDragLines	isRowExpandable	setRowHeight
disableGridLines	removeFromCell	windowInCell

IMMPlayerPanel

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

IMultiCellCanvas		
layout		

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Inherited Public Data

IMultiCellCanvas		
classDefaultStyle	dragLines	gridLines

ICanvas		
classDefaultStyle		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMMPlayerPanelHandler

IMMPlayerPanelHandler

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IHandler
    IHandler --> ICommandHandler
    ICommandHandler --> IMMPlayerPanelHandler
```

Inherited By None.

Header File immplyhd.hpp

Members

Member	Page	Member	Page
Constructor	639	rewind	641
command	640	stepBackward	641
fastForward	640	stepForward	641
pause	640	stop	641
play	640	~IMMPlayerPanelHandler	639

The IMMPlayerPanelHandler class processes the different command events for the player panel. You can override this class to provide more functionality or more precise control of the device. For example, you might call playFast instead of seekToEnd when the user presses the fast forward button for digital video. To do this, just inherit from this class and implement your own processing for fast forward.

The return value from the virtual functions specifies whether the command event is passed on for additional processing, as follows:

- true** The command event requires no additional processing. Do not pass it to another handler.
- false** The command event requires additional processing. Pass the command event on, as follows:
- If there is another handler for the window, pass the command event on to the next handler.
 - If this is the last handler for the window, call IWindow::defaultProcedure (Vol. II) to process the command event.



The multimedia interface classes are not supported on the Win32s platform.

IMMPlayerPanelHandler

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMPlayerPanelHandler

The User Interface Class Library provides only the default constructor for you to create a player panel handler object.

IMMPlayerPanelHandler();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IMMPlayerPanelHandler

virtual
~IMMPlayerPanelHandler();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process. If it is, this function calls the virtual function used to process the event.

IMMPlayerPanelHandler

command Calls the virtual functions to process a command event for a player panel.

```
virtual Boolean  
    command( ICommandEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Event Processing

Event-processing members are invoked in response to the user pressing a button on the player panel, or they are invoked by the user of this class. These can be overridden to provide more functionality or more precise control of the device. All of these methods can be thought of as button presses, even though they might have been invoked directly.

fastForward Gets called when the user presses the fast forward button. The default action is to call seekToEnd (p. 626) on the device. It also unlatches the play and pause buttons and disables the stop and pause buttons.

```
virtual Boolean  
    fastForward( const IMMPlayerPanel& panel );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	There must be a device set for the player panel.

pause Gets called when the user presses the pause button. The default action is to call pause (p. 623) on the device. If the device is already paused, it calls resume (p. 623) on the device and latches the play button.

```
virtual Boolean  
    pause( const IMMPlayerPanel& panel );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	There must be a device set for the player panel.

play Gets called when the user presses the play button. The default action is to call play (p. 624) on the device. It also unlatches the pause button and enables the stop and pause buttons.

```
virtual Boolean  
    play( const IMMPlayerPanel& panel );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	There must be a device set for the player panel.

IMMPlayerPanelHandler

rewind

Gets called when the user presses the rewind button. The default action is to call seekToStart (p. 626) on the device. It also unlatches the play and pause buttons and disables the stop and pause buttons.

```
virtual Boolean                                Win PM Motif  
    rewind( const IMMPlayerPanel& panel );      Y   Y   N
```

Exceptions	
InvalidRequest	There must be a device set for the player panel.

stepBackward

Gets called when the user presses the step backward button. This is only possible when step is meaningful to the device, such as for video. The default action is to call stepFrame (p. 627) on the device. It also unlatches the play button.

```
virtual Boolean                                Win PM Motif  
    stepBackward( const IMMPlayerPanel& panel );  Y   Y   N
```

Exceptions	
InvalidRequest	There must be a device set for the player panel.

stepForward

Gets called when the user presses the step forward button. This is only possible when step is meaningful to the device, such as for video. The default action is to call stepFrame (p. 627) on the device. It also unlatches the play button.

```
virtual Boolean                                Win PM Motif  
    stepForward( const IMMPlayerPanel& panel );  Y   Y   N
```

Exceptions	
InvalidRequest	There must be a device set for the player panel.

stop

Gets called when the user presses the stop button. The default action is to call stop (p. 625) and seekToStart (p. 626) on the device. It also unlatches the play and pause buttons and disables the stop and pause buttons.

```
virtual Boolean                                Win PM Motif  
    stop( const IMMPlayerPanel& panel );         Y   Y   N
```

Exceptions	
InvalidRequest	There must be a device set for the player panel.

IMMPlayerPanelHandler

Inherited Protected Functions

ICommandHandler		
command	dispatchHandlerEvent	systemCommand

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMPositionChangeEvent

Derivation

```

IBase
IVBase
  IEvent
    IMMPositionChangeEvent
  
```

Inherited By None.

Header File immevt.hpp

Members	Member	Page	Member	Page
	Constructor	643	userParameter	644
	device	644	~IMMPositionChangeEvent	643
	position	644		

The IMMPositionChangeEvent class is the class for position change events. They get generated when you start position tracking for a device and the position of the device changes. Use startPositionTracking (p. 622) to cause these events to be generated.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMPositionChangeEvent

Although you can construct objects of this class, typically IMMDeviceHandler::dispatchHandlerEvent (p. 552) creates objects of this class from an object of the class IEvent (Vol. II) or another IMMPositionChangeEvent object.

```
IMMPositionChangeEvent( const IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

~IMMPositionChangeEvent

```
virtual
~IMMPositionChangeEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IMMPositionChangeEvent

Event Information

Use these members to return the position, device, and user parameter for the device that generated this event.

device Returns a pointer to the device whose position is changing.

```
IMMDevice*  
device() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

position Returns the position at the time the event was generated.

```
IMMTime  
position() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

userParameter

Returns the user parameter that was set at the time of this event. See IMMDevice::setUserParameter (p. 537) for more information.

```
unsigned short  
userParameter() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMRecordable

IMMRecordable

Derivation

- IBase
- IVBase
- INotifier
- IStandardNotifier
- IMMDevice
- IMMPlayableDevice
- IMMFileMedia
- IMMRecordable

Inherited By IMMConfigurableAudio

Header File immrecrd.hpp

Members

Member	Page	Member	Page
Constructor	654	paste	649
canRedo	647	record	650
canUndo	647	redo	651
copy	647	save	651
cueForRecording	648	saveAs	652
cut	648	undo	652
deleteSelection	649	~IMMRecordable	646

The IMMRecordable class is a base class and provides all of the common behavior for all recordable media devices. There are functions for cutting, copying, and pasting from a clipboard. Also, it includes delete, undo, and save functions.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

Derived classes can use these members to create objects of this class.

~IMMRecordable

virtual			
~IMMRecordable();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

IMMRecordable

Editing

Use these members to edit and record data by the device.

canRedo Returns true if the device has any actions to be redone.

```
Boolean  
    canRedo() const; Win PM Motif  
N Y N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

canUndo Returns true if the device has any actions that can be undone.

```
Boolean  
    canUndo() const; Win PM Motif  
N Y N
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

copy Copies data from the passed-in start position to the passed-in end position into the clipboard. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the data. The clipboard is cleared before the copy occurs.

```
virtual IMMRecordable&  
    copy( const IMMTime& from = IMMTime ( ), Win PM Motif  
Y Y N  
          const IMMTime& to = IMMTime ( ),  
          CallType call = wait ) const;
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMRecordable

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the data.

cueForRecording

Cues the device for recording. This function causes any up front work that can occur to occur, thereby making the start of recording quick. This function does not have to be issued before recording; however, depending on the device, it might reduce the delay associated with the record (p. 650) function. For example, a sound card might have to clear some internal buffers before it can start recording from a microphone. This function would clear out those internal buffers for the sound card.

```
virtual IMMRecordable&  
    cueForRecording( CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

cut

Cuts the data from the passed-in start position to the passed-in end position into the clipboard. If data is already in the clipboard, it is overwritten. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the data.

```
virtual IMMRecordable&  
    cut( const IMMTime& from = IMMTime ( ),  
         const IMMTime& to = IMMTime ( ),  
         CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMRecordable

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly if the <i>to</i> position is greater than the length of the data.

deleteSelection

Deletes data from the passed-in start position to the passed-in end position. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the data.

```
virtual IMMRecordable&
deleteSelection( const IMMTime& from = IMMTime ( ),           Win PM Motif
                                                         Y   Y   N
                 const IMMTime& to = IMMTime ( ),
                 CallType call = wait );
```

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly if the <i>to</i> position is greater than the length of the data.

paste

Replaces the data with data from the clipboard, from the passed-in start position to the passed-in end position. If neither *from* nor *to* is specified, it inserts the data at the current position. If *from* is omitted, it uses the start of the buffer; if *to* is omitted, it uses the end of the audio data in the buffer. If *convert* is true, it causes the audio data in the clipboard to be converted to the current audio format before being pasted; otherwise, no conversion occurs. If you are pasting audio data, and audio format for data in the clipboard is incompatible with this device's audio format, then *convert* should be set to true; otherwise, an exception occurs.

IMMRecordable

```
virtual IMMRecordable&
paste( const IMMTime& from = IMMTime ( ),
       const IMMTime& to = IMMTime ( ),
       Boolean convert = true,
       CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>from</i> position; possibly the <i>from</i> position is greater than the end position or the <i>from</i> position is greater than the <i>to</i> position.
IAccessError	Invalid <i>to</i> position; possibly the <i>to</i> position is greater than the length of the data.
IAccessError	The disk or buffer is full.
IAccessError	There is insufficient memory to perform the operation requested.
IAccessError	The clipboard format is not valid.
IAccessError	The clipboard cannot be opened.
IAccessError	No recognizable information is in the clipboard.
IAccessError	Unable to convert audio format in the clipboard to the destination audio format.

record

Starts recording at the begin location till it reaches the end location. If the *end* location is not specified, recording continues until a pause or stop occurs. If the *begin* location is not specified, recording starts at the current location. It is recommended that you temporarily acquire the device exclusively while you are recording; otherwise, the recording can become inactive if another device requests access to the device.

```
virtual IMMRecordable&
record( Boolean insert = true,
       const IMMTime& begin = IMMTime ( ),
       const IMMTime& end = IMMTime ( ),
       Boolean resumeIfPaused = true,
       CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.

IMMRecordable

Exceptions	
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Invalid <i>begin</i> position; possibly the <i>begin</i> position is greater than the length of the data or the <i>begin</i> position is greater than the <i>end</i> position.
IAccessError	Invalid <i>end</i> position; possibly if the <i>end</i> position is greater than the length of the data.
IAccessError	The disk or buffer is full.

redo

Redoes the record, cut, paste, or delete operation most recently undone by undo. The position is at the beginning of the file after a redo. You can redo up to the last save, which causes any previous actions to be cleared and cannot be redone. Not all devices support this function.

```
virtual IMMRecordable&
    redo( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
N	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Cannot redo. Redo is not possible in the current state; possibly there are no actions to be redone (that is, the file was just saved).

save

Saves the current file.

```
virtual IMMRecordable&
    save( CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.

IMMRecordable

Exceptions	
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found; possibly the current data does not have a name. Use saveAs with the file name.
IAccessError	Save to a read-only file; possibly the currently loaded file was loaded as read-only.
IAccessError	The disk or buffer is full.
IAccessError	There is insufficient memory to perform the save operation.

saveAs

Saves the current file as a different file. Any subsequent calls to the save (p. 651) function will overwrite this file.

```
virtual IMMRecordable&
    saveAs( const IString& filename,
            CallType call = nowait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The file is not found; possibly the file name string does not contain a valid file name.
IAccessError	Save to a read-only file; possibly if the currently loaded file was loaded as read-only.
IAccessError	The disk or buffer is full.
IAccessError	There is insufficient memory to perform the save operation.

undo

Undoes the operation most recently performed by cut, record, paste, or delete. After an undo, the position is at the beginning of the media. You can undo up to the last save, which causes any previous actions to be cleared and cannot be undone. Not all devices support this function.

```
virtual IMMRecordable&
    undo( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

IMMRecordable

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	Cannot undo. Undo is not possible in the current state; possibly there are no actions to be undone (that is, the file is in the state where the last change was made).

Inherited Public Functions

IMMFileMedia		
filename	load	open
isWriteable	loadOnThread	openOnThread

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave

IMMRecordable

IMMDevice		
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	


IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

Derived classes can use these members to create objects of this class.

IMMRecordable



```
IMMRecordable( const IString& deviceName,
                Boolean openNow,
                unsigned long instance,
                Boolean openShareable );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMRecordable from the following:

deviceName
You must specify what device you want to construct.

IMMRecordable

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you have to call one of the open (p. 526) functions to open the device yourself.

instance You can provide your own instance number instead of letting IMMRecordable generate one.

openShareable
If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

```
2 IMMRecordable( unsigned long deviceIdentifier,           Win PM Motif
                  const IString& newAlias = IString ( ) );   Y  Y  N
```

A derived class can construct an IMMRecordable from the following:

deviceIdentifier
The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Inherited Protected Functions

IMMFileMedia		
enableDataUpdate	open	openOnThread

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

IMMRecordable

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMMRemovableMedia



IMMRemovableMedia

Derivation

- IBase
- IVBase
- INotifier
- IStandardNotifier
- IMMDevice
- IMMPlayableDevice
- IMMRemovableMedia

Inherited By

- IMMAudioCD
- IMMCDXA

Header File immremmed.hpp

Members	Member	Page	Member	Page
	Constructor	661	notificationHandler	662
	closeDoor	658	openDoor	659
	enableNotification	660	setNotificationHandler	662
	isMediaPresent	658	unlockDoor	659
	lockDoor	658	~IMMRemovableMedia	658
	mediaLoadedId	663		

The IMMRemovableMedia class provides all of the common behavior for all removable media devices. The main purpose of this class is to provide functions for devices that have physical media that is replaceable. Some examples of devices that might inherit from this are videoDiscs, CD players, cassette players, and VCRs. There are functions for loading or ejecting media, determining if media is present, and locking and unlocking a door on the device. Derived classes can use the virtual functions mediaLoaded and mediaUnloaded for determining if the user is changing the media.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

Derived classes can use these members to create objects of this class.

IMMRemovableMedia

~IMMRemovableMedia

```
virtual  
    ~IMMRemovableMedia();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Door and Presence

Use these members to open, close, lock, and unlock the door on the removable media device.

closeDoor Retracts the tray and closes the door, if possible.

```
virtual IMMRemovableMedia&  
    closeDoor( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

isMediaPresent

Returns true if media is inserted in the device; otherwise, it returns false.

```
Boolean  
    isMediaPresent( CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

lockDoor Locks the media cover on the device, if any. This disables manual ejection of the media from the device.

```
virtual IMMRemovableMedia&  
    lockDoor( Boolean lock = true,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

IMMRemovableMedia

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

openDoor Opens the door and ejects the tray, if possible.

```
virtual IMMRemovableMedia&  
    openDoor( Boolean open = true,  
              CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

unlockDoor Unlocks the media cover on the device, if any. This enables manual ejection of the media from the device.

```
virtual IMMRemovableMedia&  
    unlockDoor( CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
N	Y	N

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.

Event Dispatching

Event-dispatching members interact with the notification handlers for this device.

IMMRemovableMedia

enableNotification

Flags this object as being ready to accept notifications. A Notifier, like IMMRemovableMedia, does not send notifications to observer objects until it is enabled.

```
virtual IMMRemovableMedia&  
    enableNotification( Boolean enabled = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

IMMRemovableMedia

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

Derived classes can use these members to create objects of this class.

IMMRemovableMedia

```
1 IMMRemovableMedia( const IString& deviceName,                Win PM Motif
                      Boolean openNow,                          Y   Y   N
                      unsigned long instance,
                      Boolean openShareable );
```

A derived class can construct an IMMRemovableMedia from the following:

deviceName

You must specify what device you want to construct.

openNow

If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you have to call one of the open (p. 526) functions to open the device yourself.

instance

You can provide your own instance number instead of letting IMMRemovableMedia generate one.

openShareable

If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

```
2 IMMRemovableMedia( unsigned long deviceIdentifier,          Win PM Motif
                      const IString& newAlias = IString ( ) );  Y   Y   N
```

IMMRemovableMedia

A derived class can construct an IMMRemovableMedia from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias

The name you can use to associate a string to the device. Optional.

Event Dispatching

Event-dispatching members interact with the notification handlers for this device.

notificationHandler

Returns a pointer to the notification handler.

```
IMMRemovableMediaNotifyHandler*  
notificationHandler() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setNotificationHandler

Sets the notification handler to the passed value.

```
IMMRemovableMedia&  
setNotificationHandler(  
    IMMRemovableMediaNotifyHandler* notifyHandler);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

IMMRemovableMedia

INotifier		
addObserver	notifyObservers	observerList

Public Data

Notification Event Descriptions

This INotificationId string is used for notifications that IMMRemovableMedia provides to its observers.

mediaLoadedId

Notification identifier provided to observers when the current media-loaded state changes. IMMRemovableMedia provides a Boolean value in the eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the media is loaded and false, if the media is unloaded.

```
static INotificationId const
    mediaLoadedId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

IMMRemovableMedia

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMRemovableMediaHandler

Derivation

```

IBase
IVBase
IHandler
IMMDeviceHandler
IMMRemovableMediaHandler
  
```

Inherited By None.

Header File immremhd.hpp

Members	Member	Page	Member	Page
	Constructor	665	mediaUnloaded	666
	dispatchHandlerEvent	667	passDevice	666
	handleEventsFor	666	stopHandlingEventsFor	666
	mediaLoaded	666	~IMMRemovableMediaHandler	666

The IMMRemovableMediaHandler class is the base handler class for removable devices. This class provides the ability for derived classes to implement their own processing when media is loaded or unloaded from a device. For example, if the user ejects the CD from a CD player, then you might want to gray out the controls of the CD player until the user inserts a new CD.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMRemovableMediaHandler

Creates IMMRemovableMediaHandler objects.

```
IMMRemovableMediaHandler();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMRemovableMediaHandler

~IMMRemovableMediaHandler

<pre>virtual ~IMMRemovableMediaHandler();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Processing

Derived classes must supply these functions to process a pass device event.

handleEventsFor

Attaches the handler to the specified IMMDevice (p. 512) object. This is overridden to force attachment of this handler to only one device.

<pre>virtual IMMRemovableMediaHandler& handleEventsFor(IMMRemovableMedia* device);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

passDevice Processes pass device events. It is implemented by derived classes.

<pre>virtual Boolean passDevice(const IMMPassDeviceEvent& event);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

stopHandlingEventsFor

Stops the handling of events for the device passed.

<pre>virtual IMMRemovableMediaHandler& stopHandlingEventsFor(IMMRemovableMedia* device);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Media Loading

Use these members to provide processing when media is either loaded or unloaded from the device. They are implemented by derived classes.

mediaLoaded

Called whenever media is loaded into the device.

<pre>virtual IMMRemovableMediaHandler& mediaLoaded();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

mediaUnloaded

Called whenever media is unloaded from the device.

IMMRemovableMediaHandler

```
virtual IMMRemovableMediaHandler&  
mediaUnloaded();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IMMDeviceHandler		
cuePoint	handleEventsFor	positionChange
deviceEvent	passDevice	stopHandlingEventsFor

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Processing

Derived classes must supply these functions to process a pass device event.

dispatchHandlerEvent

Dispatches command functions for this handler.

```
virtual Boolean  
dispatchHandlerEvent( IEvent& event );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMRemovableMediaHandler

Inherited Protected Functions

IMMDeviceHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMRemovableMediaNotifyHandler

Derivation	IBase
	IVBase
	IHandler
	IWindowNotifyHandler
	IMMDeviceNotifyHandler
	IMMRemovableMediaNotifyHandler

Inherited By None.

Header File immremnh.hpp

Members	Member	Page
	Constructor	669
	dispatchHandlerEvent	670
	~IMMRemovableMediaNotifyHandler	670

The IMMRemovableMediaNotifyHandler class processes events for all IMMRemovableMedia devices.

This class is designed to handle events that require the device to generate a notification. If notifications are enabled for this class, a notification is generated and sent to all observers when the proper conditions for the specific notification exist.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMRemovableMediaNotifyHandler

This is the default constructor and accepts no parameters.

```
IMMRemovableMediaNotifyHandler();
```

Win	PM	Motif
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMRemovableMediaNotifyHandler

~IMMRemovableMediaNotifyHandler

```
virtual  
~IMMRemovableMediaNotifyHandler();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Processing

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it.

dispatchHandlerEvent

Notifies observers if any of the following events are received:

- media loaded event
- media unloaded event

```
virtual Boolean  
dispatchHandlerEvent( IEvent& anEvent );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMRemovableMediaNotifyHandler

Inherited Protected Functions

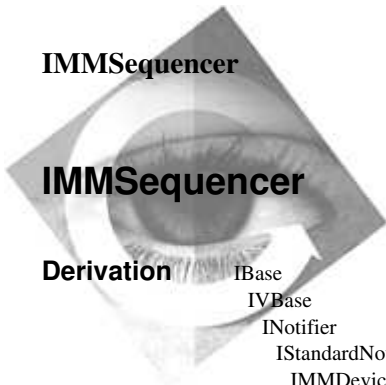
IMMDeviceNotifyHandler		
dispatchHandlerEvent	notificationHandler	setNotificationHandler

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMSequencer

IMMSequencer

Derivation

IBase
IVBase
INotifier
IStandardNotifier
IMMDevice
IMMPlayableDevice
IMMFileMedia
IMMSequencer

Inherited By None.

Header File immsequ.hpp

Members		Member	Page	Member	Page
		Constructor	672	volume	673
		isOpenStringValid	675	~IMMSequencer	673
		setVolume	673		

The IMMSequencer class is the base class for MIDI (Musical Instrument Digital Interface) playback. MIDI is a standard specification for playing back music from a series of commands, rather than from actual audio data. The commands represent musical events, such as turning a note on and off, as well as timing mechanisms for specifying the duration of the note's sound.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMSequencer

```
IMMSequencer( Boolean openNow = true,  
              unsigned long instance = 0,  
              Boolean openShareable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

You can construct an IMMSequencer from the following:

IMMSequencer

- openNow* If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.
- instance* You can provide your own instance number instead of letting IMMSequencer generate one.
- openShareable* If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

~IMMSequencer

virtual			
~IMMSequencer();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

Implementation

These members check if the passed-in string is in the correct format to open the current device.

setVolume Sets the volume of the audio channel for the device to a percent (from 0% to 100%) of the maximum audio level.

virtual IMMSequencer&			
setVolume(<u>Win</u>	<u>PM</u>	<u>Motif</u>
unsigned long volume,	Y	N	N
AudioChannel channel = all,			
const IMMILLISECONDTIME& over = IMMILLISECONDTIME (),			
CallType call = wait);			



This member is overridden in this derived class for specific operating system behavior.

volume Returns the volume of the audio channel for the device as a percent (from 0% to 100%) of the maximum audio level.

virtual unsigned long			
volume(AudioChannel channel = left,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
CallType call = wait) const;	Y	N	N



This member is overridden in this derived class for specific operating system behavior.

IMMSequencer

Inherited Public Functions

IMMFileMedia		
filename	load	open
isWriteable	loadOnThread	openOnThread

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IMMSequencer

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class.

IMMSequencer	IMMSequencer(unsigned long deviceIdentifer, const IString& newAlias = IString ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

A derived class can construct an IMMSequencer from the following:

deviceIdentifier

The value the system uses to identify the device.

newAlias The name you can use to associate a string to the device. Optional.

Implementation

These members check if the passed-in string is in the correct format to open the current device.

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

virtual Boolean	Win	PM	Motif
isOpenStringValid(const IString& deviceName) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMFileMedia		
enableDataUpdate	open	openOnThread

IMMSequencer

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMSpeed

IMMSpeed

Derivation IBase
IVBase
IMMSpeed

Inherited By None.

Header File immspeed.hpp

Members	Member	Page	Member	Page
	Constructor	677	speed	678
	format	678	~IMMSpeed	678

The IMMSpeed class provides the speed functions in frames-per-second and as a percentage. The percentage value refers to a percent of the maximum speed for the device. A case for when you might use a percentage follows. You currently are playing video at the device's fastest rate, 100 percent. You want to cut down that playback to only 60 percent of what it currently is playing. To do this, create an IMMSpeed(60) and pass it to the playAt function. If you want the video to playback at specific frames-per-second value, use the frames-per-second format.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMMSpeed

1	IMMSpeed(Format format, unsigned long speed);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
2	IMMSpeed(unsigned long percent = 100);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N

IMMSpeed

~IMMSpeed

```
virtual  
    ~IMMSpeed();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Format

Use these members to specify the speed as a percentage or in frames-per-second. These members return the current speed format and value.

format Returns the current speed format. This is either in frames-per-second or percentage.

```
Format  
    format() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

speed Returns the speed value as either a percentage or in frames-per-second.

```
virtual unsigned long  
    speed() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

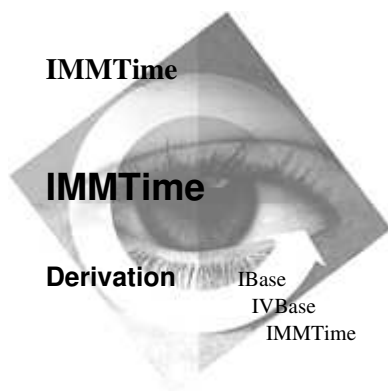
IBase		
recoverable	unrecoverable	

Format

```
Format {  
    percentage = 3,  
    framesPerSecond  
};
```


IMMSpeed

Enumeration for specifying the way the speed is determined.



Inherited By	IMMHourMinSecFrameTime	IMMMinSecFrameTime
	IMMHourMinSecTime	IMMTrackMinSecFrameTime
	IMMMillisecondTime	

Header File `immtime.hpp`

Members	Member	Page	Member	Page
	Constructor	681	operator <=	684
	asMMTime	682	operator =	682
	asString	683	operator ==	684
	defaultTime	686	operator >	685
	hours	681	operator >=	685
	hundredths	681	operator unsigned long	682
	isValid	683	ordinal	685
	minutes	681	seconds	681
	operator !=	683	setMMTime	686
	operator +	683	setTimeToOrdinal	685
	operator +=	683	setValid	686
	operator -	684	thousandths	681
	operator -=	684	~IMMTime	682
	operator <	684		

The IMMTime base device time class provides behavior common to all device times. IMMTime is the standard time and media position format supported by all of the multimedia devices. Most time values can be represented as unsigned long values. This class also provides function for converting an IMMTime value to an unsigned long value. All time values are based on 4 bytes. Because each time class is in different units, the maximum time value that each can contain is different.

PM For OS/2, this time unit is 1/3000 second, or 333 microseconds (Universal Chinatown time).

Win For Win32, this time unit is 1 millisecond.

W32s The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Attributes

Use these members to parse the time into normal time values, for example, hours, minutes, and seconds.

hours Returns the hours component of the time.

virtual unsigned long hours() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

hundredths Returns the millisecond component of the time rounded to the nearest hundredth.

virtual unsigned long hundredths() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

minutes Returns the minutes component of the time. This is in the range of 0 to 59.

virtual unsigned long minutes() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

seconds Returns the seconds component of the time. This is in the range of 0 to 59.

virtual unsigned long seconds() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

thousandths Returns the millisecond component of the time rounded to the nearest thousandth.

virtual unsigned long thousandths() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

IMMTime

1 IMMTime(unsigned long time = defaultTime);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMTime from the following:

IMMTime

time A time value where each unit is equal to the standard time unit for this platform.

PM For OS/2, this time unit is 1/3000 second, or 333 microseconds (Universal Chinatown time).

Win For Win32, this time unit is 1 millisecond.

2 IMMTime(const IMMTime& time);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

operator = Sets the time to be equal to the passed-in time.

IMMTime&
operator =(const IMMTime& time);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed in time was invalid.

~IMMTime

virtual
~IMMTime();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversions

Use these members to cast the time to an unsigned long or an unsigned long that is in the IMMTime format.

asMMTime Returns this time value as an unsigned long that can be used to create an IMMTime (p. 680). By default, this returns this time value. Derived classes should override this function to provide a way to be converted to an IMMTime value.

virtual unsigned long
asMMTime() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

operator unsigned long

Returns the time as a unsigned long where each time unit is equal to 1/3000 of a second.

virtual
operator unsigned long() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMTime

Diagnostics

Use these members to return the time as a string and allow you to determine if the time is valid.

asString Returns the time value as a string formatted as HH:MM:SS.MMM.

virtual IString asString() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--------------------------------------	------------------------	-----------------------	--------------------------

isValid Returns true if the time value is valid.

virtual Boolean isValid() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
-------------------------------------	------------------------	-----------------------	--------------------------

Operators

Use these operators to perform different mathematical operations on this time object.

operator != Returns true if this time is not equal to the passed-in time.

Boolean operator !=(const IMMTime& time) const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	------------------------	-----------------------	--------------------------

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator + Returns the sum of this time and the passed-in time.

IMMTime operator +(const IMMTime& time) const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator += Adds the passed-in *time* from this time object

IMMTime& operator +=(const IMMTime& time);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	------------------------	-----------------------	--------------------------

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

IMMTime

operator - Returns the result of this time minus the passed-in time.

IMMTime	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator -(const IMMTime& time) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator -= Subtracts the passed-in *time* from this time object.

IMMTime&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator -=(const IMMTime& time);	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator < Returns true if this time is less than the passed-in time.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator <(const IMMTime& time) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator <= Returns true if this time is less than or equal to the passed-in time.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator <=(const IMMTime& time) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator == Returns true if this time is equal to the passed-in time.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IMMTime& time) const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

IMMTime

operator > Returns true if this time is greater than the passed-in time.

Boolean
operator >(const IMMTime& time) const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

operator >= Returns true if this time is greater than or equal to the passed-in time.

Boolean
operator >=(const IMMTime& time) const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in time is invalid or this time object is invalid.

Ordinal

Use these members to indicate generic positions. A time object returns an ordinal number, which represents the time at some particular granularity. This is useful for something that allows seeking and displays position, such as a slider. Because sliders generally do not process time values, use an ordinal number to establish the range of slider values. Ordinal numbers can be mathematically manipulated and converted back to time values, if necessary.

Note: Some time values are difficult to represent as an ordinal value, such as IMMTrackMinSecFrameTime. Such a time object produces ordinal numbers that are dependent on unknown information, such as the table of contents of a CD. Some time classes might not be able to correctly implement setTimeToOrdinal (p. 685).

ordinal Returns an ordinal number in milliseconds. By default, this is the time converted to milliseconds. Not all derived classes can return milliseconds.

virtual unsigned long
ordinal() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setTimeToOrdinal

Sets the time object to the value represented by the ordinal number in milliseconds.

virtual IMMTime&
setTimeToOrdinal(unsigned long ordinal);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IMMTime

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Implementation

These members set the validity of the time and change the time to a new value.

setMMTime Sets the current time to the new value.

IMMTime&
setMMTime(unsigned long newTime);

Win

PM

Motif

Y

Y

N

PM

For OS/2, the time units are 1/3000 second, or 333 microseconds (Universal Chinatown time).

Win

For Win32, the time unit is 1 millisecond.

setValid Sets the validity of the current time value. With setValid, derived classes can change the validity of the time value. Validity means that this time object represents a valid time.

IMMTime&
setValid(Boolean Valid = true);

Win

PM

Motif

Y

Y

N

Public Data

Constructors

You can construct and destruct objects of this class.

defaultTime Used to denote a default time.

686 Open Class Library Reference

IMMTime

static const unsigned long
defaultTime;

WinPMMotif
*Y**Y**N*

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Format

Format {
 milliseconds = 1,
 trackMinSecFrame,
 bytes = 11,
 mm24FramesPerSecond,
 mm30FramesPerSecondDrop,
};

mmTime,
frames = 8,
samples,
mm25FramesPerSecond,
songpointer

minSecFrame = 5,
hourMinSec,
hourMinSecFrame,
mm30FramesPerSecond,

This enumeration denotes the different allowable time formats. Valid values are as follows:

milliseconds
Represents the 1/1000 of a second time format.

mmTime
Represents the 1/3000 of a second time format.

minSecFrame
Represents the time format based on the 75-frames-per-second CD digital audio standard (another name for this is Redbook time format).

trackMinSecFrame
Represents the tracks-minutes-seconds-frames time format based on the 75-frames-per-second CD digital audio standard. This format is used primarily by CD audio devices.

frames
Represents the digital video frames time, where a number corresponds to a frame number.

hourMinSec
Represents the hours-minutes-seconds time format commonly used by videoDisc and digital video players.

bytes
Represents the wave audio bytes time, where a number corresponds to a byte number.

© IBM Corp. 1992, 1996

IMMTime 687

IMMTime

samples

Represents the wave audio samples time, where a number corresponds to a sample number.

hourMinSecFrame

Represents the hours-minutes-seconds-frames time commonly used by videoDisc and digital video players.

mm24FramesPerSecond

Represents the frame-numbering system developed by the Society of Motion Picture and Television Engineers that assigns a number to each frame of video based on 24 frames-per-second.

mm25FramesPerSecond

Represents the frame-numbering system developed by the Society of Motion Picture and Television Engineers that assigns a number to each frame of video based on 25 frames-per-second.

mm30FramesPerSecond

Represents the frame-numbering system developed by the Society of Motion Picture and Television Engineers that assigns a number to each frame of video based on 30 frames-per-second.

30FramesPerSecondDrop

Represents the frame-numbering system developed by the Society of Motion Picture and Television Engineers that assigns a number to each frame of video based on 30 frames-per-second.

songpointer

Represents sixteenth notes. This is the default time format for some MIDI devices.



The Win32 MCI subsystem does not support Universal Chinatown time (1/3000 second). Therefore, any references to the mmTime enumeration will be interpreted as milliseconds.



IMMTrackMinSecFrameTime

Derivation

```

IBase
  IVBase
    IMMTime
      IMMTrackMinSecFrameTime
  
```

Inherited By None.

Header File immtime.hpp

Members	Member	Page	Member	Page
	Constructor	690	operator -=	692
	asString	692	operator =	691
	frames	689	operator unsigned long	691
	minutes	689	seconds	690
	operator +	692	track	690
	operator +=	692	~IMMTrackMinSecFrameTime	691
	operator -	692		

The IMMTrackMinSecFrameTime data type class represents the tracks-minutes-seconds-frames (TTMMSSFF) time format. This format is used primarily by compact disc audio devices.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Attributes

Use these members to parse the data into tracks, seconds, minutes, and frames.

frames Returns the frames component of the time.

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
frames() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

minutes Returns the minutes component of the time. This is in the range of 0 to 59.

virtual unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minutes() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

IMMTrackMinSecFrameTime

seconds Returns the seconds component of the time. This is in the range of 0 to 59.

```
virtual unsigned long
seconds() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

track Returns the track component of the time.

```
virtual unsigned long
track() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct, copy, and destruct objects of this class.

IMMTrackMinSecFrameTime

```
1 IMMTrackMinSecFrameTime( IMMAudioCDContents& contents,
                           unsigned long track,
                           unsigned long minutes,
                           unsigned long seconds,
                           unsigned long frames );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMHourMinSecFrameTime from the following:

- contents* A table of contents for the compact disc that you want to create a time for.
- track* A track number.
- minutes* The number of minutes.
- seconds* The number of seconds.
- frames* The number of frames.

```
2 IMMTrackMinSecFrameTime( const IMMTrackMinSecFrameTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

```
3 IMMTrackMinSecFrameTime( IMMAudioCDContents& contents,
                           unsigned long value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct an IMMTrackMinSecFrameTime from the following:

IMMTrackMinSecFrameTime

contents A table of contents for the compact disc that you want to create a time for.

value A time value where:

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the track

operator = Sets the time to be equal to the passed-in time.

```
IMMTrackMinSecFrameTime&  
operator =( const IMMTrackMinSecFrameTime& time );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The passed-in <i>time</i> is invalid.

~IMMTrackMinSecFrameTime

```
virtual  
~IMMTrackMinSecFrameTime();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Conversion

Use these members to cast the time to an unsigned long.

operator unsigned long

Returns the time as an unsigned long in the following format (FFSSMMTT):

1. 1st byte is the frames
2. 2nd byte is the seconds
3. 3rd byte is the minutes
4. 4th byte is the tracks

This value is based on the current table of contents.

```
virtual  
operator unsigned long() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Diagnostics

Use these members to return the time as a string.

IMMTrackMinSecFrameTime

asString Returns the time value as a string formatted as TT MM:SS.FF.

<pre>virtual IString asString() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Operators

Use these operators to perform mathematical operations on time objects.

operator + Returns the sum of the current set time and the passed-in time.

<pre>IMMTrackMinSecFrameTime operator +(const IMMTime& time);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
InvalidRequest	The passed-in <i>time</i> is invalid or this time object is invalid.

operator += Adds the *time* to the current set time.

<pre>IMMTrackMinSecFrameTime& operator +=(const IMMTime& time);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
InvalidRequest	The passed-in <i>time</i> is invalid or this time object is invalid.

operator - Returns the result of the current set time minus the passed-in time.

<pre>IMMTrackMinSecFrameTime operator -(const IMMTime& time);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
InvalidRequest	The passed-in <i>time</i> is invalid or this time object is invalid.

operator -= Subtracts the *time* from the current set time.

<pre>IMMTrackMinSecFrameTime& operator -=(const IMMTime& time);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Exceptions	
InvalidRequest	The passed-in <i>time</i> is invalid or this time object is invalid.

IMMTrackMinSecFrameTime

Inherited Public Functions

IMMTime		
asMMTime	operator +	operator ==
asString	operator +=	operator >
hours	operator -	operator >=
hundredths	operator -=	operator unsigned long
isValid	operator <	ordinal
minutes	operator <=	seconds
operator !=	operator =	setTimeToOrdinal

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IMMTime		
setMMTime	setValid	

Inherited Public Data

IMMTime		
defaultTime		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMMWaveAudio

IMMWaveAudio

Derivation

IBase
IVBase
INotifier
IStandardNotifier
IMMDevice
IMMPlayableDevice
IMMFileMedia
IMMRecordable
IMMConfigurableAudio
IMMWaveAudio

Inherited By None.

Header File immwave.hpp

Members

Member	Page	Member	Page
Constructor	698	isOpenStringValid	701
copyFromBuffer	694	pasteFromBuffer	696
copyToBuffer	695	pasteToBuffer	697
cutCopyBufferSize	695	supportsWaveFormat	697
cutToBuffer	696	~IMMWaveAudio	698

The IMMWaveAudio class provides recording and playing of sound files. The sound files are stored in a waveform format. Not only can you use the inherited functions, such as play, pause, and setFormat, but you can also use this class for cutting, copying, and pasting to and from an audio buffer.



The multimedia interface classes are not supported on the Win32s platform.

Public Functions

Buffers

Use these members to edit the audio using buffers and setting attributes for the current audio.

copyFromBuffer

Copies all of the audio data from the audio buffer to the current location.

IMMWaveAudio

```
virtual IMMWaveAudio&
    copyFromBuffer( const IMMAudioBuffer& audioBuffer,
                    CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.

copyToBuffer

Copies audio data into the audio buffer from the passed-in begin position to the passed-in end position. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the audio data.

```
virtual IMMWaveAudio&
    copyToBuffer( IMMAudioBuffer& audioBuffer,
                  const IMMTime& from = IMMTime ( ),
                  const IMMTime& to = IMMTime ( ),
                  CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The buffer is too small to hold that data.
IAccessError	The passed-in range is invalid; either the <i>to</i> or the <i>from</i> position does not point to a valid position or the difference between <i>from</i> and <i>to</i> is not greater than zero.

cutCopyBufferSize

Returns the size the audio buffer needs to be to contain the audio data from the passed-in beginning position to the passed-in ending position. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the audio data.

```
virtual unsigned long
    cutCopyBufferSize( const IMMTime& from = IMMTime ( ),
                       const IMMTime& to = IMMTime ( ) ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

IMMWaveAudio

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The buffer is too small to hold that data.
IAccessError	The passed-in range is invalid; either the <i>to</i> or the <i>from</i> position does not point to a valid position or the difference between <i>from</i> and <i>to</i> is not greater than zero.

cutToBuffer Cuts audio data into the audio buffer from the passed-in begin position to the passed-in end position. If *from* is omitted, it uses the current position; if *to* is omitted, it uses the end of the audio data.

```
virtual IMMWaveAudio&
    cutToBuffer( IMMAudioBuffer& audioBuffer,
                const IMMTime& from = IMMTime ( ),
                const IMMTime& to = IMMTime ( ),
                CallType call = wait );
```

Win	PM	Motif
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The buffer is too small to hold that data.
IAccessError	The passed-in range is invalid; either the <i>to</i> or the <i>from</i> position does not point to a valid position or the difference between <i>from</i> and <i>to</i> is not greater than zero.

pasteFromBuffer

Replaces the audio data with data from the audio buffer, from the passed-in begin position to the passed-in end position. If neither *from* nor *to* is specified, then it inserts the data at the current position. If *from* is omitted, it uses the start of the buffer; if *to* is omitted, it uses the end of the audio data in the buffer.

IMMWaveAudio

```
virtual IMMWaveAudio&
pasteFromBuffer( const IMMAudioBuffer& audioBuffer,
                 const IMMTime& from = IMMTime ( ),
                 const IMMTime& to = IMMTime ( ),
                 CallType call = wait );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
IAccessError	The passed-in range is invalid; either the <i>to</i> or the <i>from</i> position does not point to a valid position or the difference between <i>from</i> and <i>to</i> is not greater than zero.

pasteToBuffer

Pastes audio data into the passed-in audio buffer.

```
virtual IMMWaveAudio&
pasteToBuffer( IMMAudioBuffer& audioBuffer,
               CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
InvalidRequest	The device must be in the open state before calling this function.
IAccessError	The buffer is too small to hold that data.

supportsWaveFormat

Returns true if the format is supported. This allows an application to determine whether a specific wave audio format is supported.

```
virtual Boolean
supportsWaveFormat( unsigned long bitsPerSample,
                    unsigned long samplesPerSecond,
                    unsigned long channels,
                    IMMAudioBuffer::Format format,
                    CallType call = wait ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>Y</i>	<i>N</i>

IMMWaveAudio

Exceptions	
IAccessError	The device identifier is not valid; possibly the device is closed.
IAccessError	The device cannot acquire access to the hardware device; possibly another device of this same type was acquired for exclusive use in this or another process.
IAccessError	An internal driver error occurred.
IInvalidRequest	The device must be in the open state before calling this function.
IAccessError	The device hardware does not support this format; possibly the combination of the passed-in audio attributes is not supported by the system.

Constructors

You can construct and destruct objects of this class.

IMMWaveAudio

```
IMMWaveAudio( Boolean openNow = true,           Win PM Motif
               unsigned long instance = 0,       Y   Y   N
               Boolean openShareable = true );
```

You can construct an IMMWaveAudio from the following:

openNow If true, it causes the device to automatically open the device before returning from the constructor; otherwise, you would have to call one of the open (p. 526) functions to open the device yourself.

instance You can provide your own instance number instead of letting IMMWaveAudio generate one.

openShareable
If true, it allows the hardware device to be shared by different programs; otherwise, the hardware cannot be shared.

~IMMWaveAudio

```
virtual                                           Win PM Motif
~IMMWaveAudio();                               Y   Y   N
```

Inherited Public Functions

IMMWaveAudio

IMMConfigurableAudio		
bitsPerSample	format	setBytesPerSecond
blockAlignment	samplesPerSecond	setChannels
bytesPerSecond	setBitsPerSample	setFormat
channels	setBlockAlignment	setSamplesPerSecond

IMMRecordable		
canRedo	cut	redo
canUndo	deleteSelection	save
copy	paste	saveAs
cueForRecording	record	undo

IMMFileMedia		
filename	load	open
isWriteable	loadOnThread	openOnThread

IMMPlayableDevice		
addCuePoint	position	seekToStart
cueForPlayback	removeCuePoint	startPositionTracking
length	resume	stepFrame
pause	seek	stop
play	seekToEnd	stopPositionTracking

IMMDevice		
acquire	isAudioEnabled	setVolume
aliasName	isCloseOnDestroy	speedFormat
close	isConnectionSupported	supportsAudio
connectedDeviceId	isConnectorEnabled	supportsCommand
deletePendingEvents	isOpen	supportsDigitalTransfer
description	mode	supportsDisableEject
deviceId	open	supportsEject
deviceName	openOnThread	supportsPlay
deviceType	prerollTime	supportsRecord
disableAudio	prerollType	supportsRecordInsertion
disableConnector	release	supportsSave

IMMWaveAudio

IMMDevice		
enableAudio	requiresFiles	supportsStreaming
enableConnector	setCloseOnDestroy	supportsVideo
enableNotification	setSpeedFormat	supportsVolumeAdjustment
isAcquired	setTimeFormat	timeFormat

IStandardNotifier		
disableNotification	enableNotification	isEnabledForNotification

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class.

IMMWaveAudio

IMMWaveAudio(unsigned long deviceIdentifier,
const IString& newAlias = IString ());

Win

PM

Motif

Y

Y

N

A derived class can construct an IMMWaveAudio from the following:

- deviceIdentifier

The value the system uses to identify the device.
- newAlias

The name you can use to associate a string to the device. Optional.

IMMWaveAudio

Implementation

These members check if the passed-in string is in the correct format to open the current device.

isOpenStringValid

Returns true if the passed-in open string is valid for this device.

```
virtual Boolean  
    isOpenStringValid( const IString& deviceName ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IMMConfigurableAudio		
translateAudioFlag		

IMMFileMedia		
enableDataUpdate	open	openOnThread

IMMDevice		
deviceWindow	open	setOpenStatus
isOpenStringValid	openOnThread	setPassDeviceRequested
itemCapability	openStatus	setSystemMixerHandle
itemStatus	sendCommand	setUserParameter
lastError	setLastError	systemMixerHandle
mixerControlValues	setMixerControlValues	userParameter
notificationHandler	setNotificationHandler	wasPassDeviceRequested

IStandardNotifier		
addObserver	notifyObservers	observerList

INotifier		
addObserver	notifyObservers	observerList

IMMWaveAudio

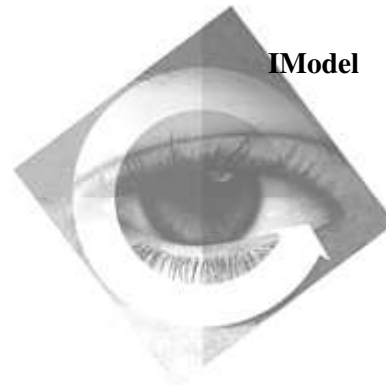
Inherited Public Data

IMMDevice		
allDevices	cuePointId	other
ampMixer	dat	overlay
animation	deviceEventId	passDeviceId
audioCD	digitalVideo	positionChangeId
audioTape	headphone	sequencer
cdxa	microphone	speaker
commandNotifyId	monitor	videoDisc

IStandardNotifier		
deleteId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IModel

Derivation

IBase
IVBase
IModel

Inherited By

IEmbedderModel

Header File

imodel.hpp

Members

Member	Page	Member	Page
Constructor	707	operator <<=	706
attachTo	706	operator >>=	706
component	705	operator INotifier &	705
hasChanged	704	setChanged	705
notifier	704	~IModel	706
notifyOfChange	704		

Use the IModel class to provide a model for a server. A model contains the server component's data, which can be text, graphics, video, sound, or any other type of data.

Servers that derive their models from IModel can be embedded, but they cannot have objects embedded in them. For example, paintbrush applications are typically servers since pictures are inserted into spreadsheets and word processors, but objects are never inserted inside a paintbrush picture.

Create your own derived class of IModel to create the models for servers in your application.

To create models for containers, create a subclass that derives from IEmbedderModel (p. 243). Container components have all the behavior of server components, and in addition they can have components embedded in them.

IModel objects are stored by default in flat file storage, through the IFlatFileStorage class. Structured storage is not required because a server does not need to store any embedded components. However, if you want to create a server that uses structured storage, you can accomplish this by overriding the IComponent::isStructuredStorage function (p. 10) in your subclass to return true. Objects of your derived class will then be saved to structured storage using the IStructuredStorage class.

IModel

See IFlatFileStorage (p. 250) and IStructuredStorage (p. 711) for more information on the storage classes.

Public Functions

Change Notification

Use these members to obtain notification of changes to a model, and to control the way the framework is notified of changes.

hasChanged Returns whether or not the model (or any of its related data objects) has changed.

Called by the framework when the end-user selects options such as **File ► Exit** or **File ► Save**, to determine if the model object needs to be streamed out. Model changes are indicated using the setChanged function.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasChanged() const;	<i>Y</i>	<i>N</i>	<i>N</i>

notifier Returns the document notifier, which is used for change notification. The actual notifier object is maintained by class IComponent, allowing models to be replaced independently of, and without re-registration of, the observers (for example, views).

See INotifier (Vol. I) for more information.

virtual INotifier&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
notifier() const;	<i>Y</i>	<i>N</i>	<i>N</i>

notifyOfChange

Sends the specified notification to interested parties (such as views and containers). Objects that have registered with the model notifier (returned by IModel::notifier) receive a complete notification, including, the specified notification event. (Views are automatically registered with the model notifier by the framework.)

Containers that are not explicitly registered as observers only receive a generic notification that a change has occurred, without receiving the specific notification event.

Note: If the supplied notification event indicates an attribute change notification (event.notifierAttrChanged() == true), this function will also call setChanged(true). Otherwise, no call to setChanged is made, and only the notification is sent. Because notification events by default represent attribute changes, notifyOfChange will by default also call setChanged. See class INotificationEvent for more details.

IModel

You should generally call this method in your model derived class, wherever changes to the model are made.

See INotificationEvent (Vol. I) for more information about notification events.

virtual void notifyOfChange(const INotificationEvent&);	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
--	-----------------	----------------	-------------------

The parameter for this function is as follows:

INotificationEvent&

A notification event which provides the details of an event to an observer.

operator INotifier &

This convenience operator simply calls IModel::notifier() and enables automatic casting of model objects to INotifier objects. You can use the operator to pass a model object wherever a notifier object is required, as when you are constructing INotificationEvent objects to be passed to IModel::notifyOfChange.

See INotifier (Vol. I) for more information about notification protocols.

operator INotifier &() const;	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
-------------------------------	-----------------	----------------	-------------------

setChanged

Sets or resets the changed value for this model, as indicated by the supplied boolean argument. If the parameter is empty, it defaults to true, indicating that the document has changed. Passing a false value to this function prevents the framework from saving a model in which changes have been made. Note however that this call will not undo the actual change in the model object.

Clients generally do not call setChanged(true) directly, but instead call IModel::notifyOfChange, which also notifies any registered observers.

virtual void setChanged(Boolean changed = true);	<u>Win</u> Y	<u>PM</u> N	<u>Motif</u> N
---	-----------------	----------------	-------------------

Informational

Use this member to obtain a reference to the component to which the model belongs.

component

Returns the component to which the model belongs. The component is associated with the model using the IModel::attachTo function, which is called by the framework immediately after the model is created.

IModel

```
IComponent&  
    component() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Initialization

The framework uses this member to attach the model to a component.

attachTo

Attaches the model to a component. This function is called by IComponent::adoptModel (p. 8), when the model is adopted by a component.

When the framework constructs a model object, the framework immediately arranges for a component to adopt the model. Models cannot function in the framework before they are adopted by a component.

See IComponent (p. 7) for more information about components.

```
virtual void  
    attachTo( IComponent& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

IComponent&
A component.

Constructors

You can destruct objects of this class. You cannot construct objects of this class directly (the default constructor is protected). In the current release, you cannot copy objects of this class.

~IModel

```
virtual  
    ~IModel();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Streaming

Use these members to stream objects of this class in and out.

operator <<= The stream-in operator for objects of this class.

```
virtual IBaseStream&  
    operator <<=( IBaseStream& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

operator >>= The stream-out operator for objects of this class.

IModel

```
virtual IBaseStream&  
operator >>= ( IBaseStream& ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can destruct objects of this class. You cannot construct objects of this class directly (the default constructor is protected). In the current release, you cannot copy objects of this class.

IModel

1 IModel ();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Default constructor. Model objects are constructed by the framework's factory function IComponentStationery::createModel (p. 15). IModel objects are never instantiated directly. Instead, you must provide a model subclass. The model subclass's constructor must explicitly or implicitly call this IModel default constructor.

2 IModel(const IModel&);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

Do **not** use this function. Copying of models is not supported in the current release. This function only exists to support the extended type system and is otherwise unused. Calls to this function directly, or using the dynamic copy function ::copy(), result in a run-time assertion.

IModel

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IRegionHandle

Derivation IBase
 IHandle
 IRegionHandle


Inherited By None.

Header File ihandle.hpp

Members	Member	Page
	Constructor	709
	operator Value	709

The IRegionHandle class accesses a window's update region.

 IRegionHandle is an alias for the OS/2 Programmer's Toolkit HRGN typedef.

 The AIX release does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IRegionHandle

You can construct objects of this class from a region handle (a value of type IHandle::Value), which defaults to 0.

```
IRegionHandle( Value hregion = 0 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Operators

This group contains operators for this class.

operator Value Returns the IHandle value.

IRegionHandle

operator Value() const;

Win

PM

Motif

N

N

Y

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	unrecoverable	



IStructuredStorage

Derivation

```

IBase
IVBase
IDocumentStorage
IStructuredStorage
  
```

Inherited By None.

Header File idocstor.hpp

Members	Member	Page	Member	Page
	Constructor	712	saveFile	712
	loadFile	712	~IStructuredStorage	712

The IStructuredStorage class stores the document model in an OLE Compound Document file (formerly known as a DocFile). This class **must** be used for storing container components (components that support embedding). Container components derive from IEmbedderModel (p. 243).

You can also use this class to store non-container components, although the framework defaults to flat file storage for non-containers. Non-containers derive from IModel (p. 703), and use storage derived from IFlatFileStorage (p. 250). To change the default and save a non-container to structured storage, override the IComponent::isStructuredStorage (p. 10) function to return true.

An OLE Compound File manages a compound document using OLE storages and streams, that contain the model data for a container component and its embedded components. The streams and storages can be thought of as files and directories in a file system:

- A storage can hold streams and other storages, just as a directory can hold files and other directories.
- A stream holds actual data.

Public Functions

Constructors

The constructors and destructors for this class are called by the framework. You do not need to manage storage objects directly.

IStructuredStorage

IStructuredStorage

IStructuredStorage(IComponent&,
 IEmbedderModel*);

Win

PM

Motif

Y

N

N

This constructor is called by the framework to create a storage object for components that use structured storage (for example, container components). You do not need to explicitly instantiate storage objects.

Exceptions	
IAccessError	Failed to initialize the underlying compound document storage system. Unrecoverable.

~IStructuredStorage

virtual
~IStructuredStorage();

Win

PM

Motif

Y

N

N

File I/O

Use these members to save or load a model to or from a file.

loadFile

Loads the model data for a component from an OLE Compound file (formerly known as a DocFile). If used on a container, this function also loads any embedded components. This is a concrete implementation of IDocumentStorage::loadFile (p. 231).

virtual void
loadFile(const IString& name);

Win

PM

Motif

Y

N

N

The parameter for this function is as follows:

IString& The name of the file to be loaded.

Exceptions	
IAccessError	Unexpected read error. Recoverable. This function also passes through many other stream-in and meta type system exceptions, for example, "invalid stream", "unsupported version".

saveFile

Saves the model data for a component to an OLE Compound file (formerly known as a DocFile). If used on a container, this function also saves any embedded components. This is a concrete implementation of IDocumentStorage::saveFile (p. 232).

IStructuredStorage

```
virtual void
    saveFile( const IString& name,
              Boolean sameAsLoad,
              Boolean remember );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

- IString&* The name of the file to be saved.
- Boolean* Indicates whether the name being used is the same as that used during loading (**File ► Save**), or not (**File ► Save As** and **File ► Save Copy As**).
- Boolean* Indicates that the document's file name should be changed to the supplied name.

Exceptions	
IAccessError	Unexpected write error. Recoverable.

Inherited Public Functions

IDocumentStorage		
component	fileName	loadFile

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

The constructors and destructors for this class are called by the framework. You do not need to manage storage objects directly.

```
IStructuredStorage IStructuredStorage( const IStructuredStorage& );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>N</i>	<i>N</i>

The copy constructor. You cannot copy objects of this class directly.

IStructuredStorage

Exceptions	
IAccessError	Failed to initialize the underlying compound document storage system. Unrecoverable.

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ITransformMatrix

Derivation IBase
ITransformMatrix

Inherited By None.

Header File itrnsfrm.hpp

Members	Member	Page	Member	Page
	Constructor	716	rotateBy	719
	asMATRIXLF	717	scaleBy	720
	asTagXFORM	717	setElement11	718
	element11	718	setElement12	719
	element12	718	setElement21	719
	element21	718	setElement22	719
	element22	718	setElement31	719
	element31	718	setElement32	719
	element32	718	setToIdentity	719
	operator !=	715	translateBy	720
	operator =	717	~ITransformMatrix	717
	operator ==	715		

The ITransformMatrix class is used to represent a 3x3 transformation matrix.

You can use ITransformMatrix objects to quickly construct transformation matrixes for use with IGraphic::setTransformMatrix or with the native graphic programming interface.

Public Functions

Comparisons

Use these members to compare two ITransformMatrix objects.

operator != Returns true if the transform matrixes are not equal.

IBase::Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator !=(const ITransformMatrix& transformMatrix) const;	Y	Y	N

operator == Returns true if the transform matrixes are equal.

ITransformMatrix

```
IBase::Boolean
operator ==( const ITransformMatrix& transformMatrix ) const; Win PM Motif
Y Y N
```

Constructors

You can construct, destruct, copy, and assign objects of this class.

ITransformMatrix

```
1 ITransformMatrix(); Win PM Motif
Y Y N
```

Use the default construct to create an ITransformMatrix object that is initialized to the identity matrix.

```
2 ITransformMatrix( double element11, Win PM Motif
double element12, Y Y N
double element21,
double element22,
long element31,
long element32 );
```

Use this function to construct an ITransformMatrix object from six of the nine elements in the matrix. The elements at 1,3, 2,3, and 3,3 default to 0, 0, and 1 respectively.

- element11* Matrix element 1,1 (Used to scale, rotate, and reflect).
- element12* Matrix element 1,2 (Used to scale and rotate).
- element21* Matrix element 2,1 (Used to scale and rotate).
- element22* Matrix element 2,2 (Used to scale, rotate, and reflect).
- element31* Matrix element 3,1 (Used to translate).
- element32* Matrix element 3,2 (Used to translate).

```
3 ITransformMatrix( const struct _MATRIXLF& matrixlf ); Win PM Motif
N Y N
```

Use this function to construct an ITransformMatrix object from a PM Toolkit MATRIXLF structure.

- matrixlf* Presentation Manager MATRIXLF structure.

ITransformMatrix

4 ITransformMatrix(const struct tagXFORM& tagxform); Win PM Motif
Y N N

Use this function to construct an ITransformMatrix object from a Windows Toolkit tagXFORM structure.

matrixlf Windows tagXFORM structure.

5 ITransformMatrix(const ITransformMatrix& transformMatrix); Win PM Motif
Y Y N

Use this function to construct an ITransformMatrix object from another ITransformMatrix object.

transformMatrix A ITransformMatrix object.

operator = Use this function to assign a ITransformMatrix object to another ITransformMatrix object.

ITransformMatrix&
operator =(const ITransformMatrix& matrix); Win PM Motif
Y Y N

~ITransformMatrix

~ITransformMatrix(); Win PM Motif
Y Y N

Matrix Elements

Use these members to set and query the elements of the transform matrix. In addition, there is a member that resets the transform matrix to the identity matrix (setToIdentity) and one that returns the transform matrix as an environment-specific structure (asMATRIXLF in the OS/2 operating system and asTagXFORM in Windows).

asMATRIXLF Renders the matrix as an OS/2 Developers Toolkit MATRIXLF structure.

struct _MATRIXLF
asMATRIXLF() const; Win PM Motif
N Y N

asTagXFORM

Renders the matrix as a Win32 tagXFORM structure.

ITransformMatrix

	<pre>struct tagXFORM asTagXFORM() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
element11	Returns the value for the matrix element 1,1.			
	<pre>double element11() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
element12	Returns the value for the matrix element 1,2.			
	<pre>double element12() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
element21	Returns the value for the matrix element 2,1.			
	<pre>double element21() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
element22	Returns the value for the matrix element 2,2.			
	<pre>double element22() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
element31	Returns the value for the matrix element 3,1.			
	<pre>long element31() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
element32	Returns the value for the matrix element 3,2.			
	<pre>long element32() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
setElement11	Sets the value for element 1,1 in the transformation matrix.			
	<pre>ITransformMatrix& setElement11(double value);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>

ITransformMatrix

setElement12

Sets the value for element 1,2 in the transformation matrix.

```
ITransformMatrix&  
    setElement12( double value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setElement21

Sets the value for element 2,2 in the transformation matrix.

```
ITransformMatrix&  
    setElement21( double value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setElement22

Sets the value for element 2,2 in the transformation matrix.

```
ITransformMatrix&  
    setElement22( double value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setElement31

Sets the value for element 3,1 in the transformation matrix.

```
ITransformMatrix&  
    setElement31( long value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setElement32

Sets the value for element 3,2 in the transformation matrix.

```
ITransformMatrix&  
    setElement32( long value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setToIdentity Resets the transformation matrix to the identity matrix.

```
ITransformMatrix&  
    setToIdentity();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Scale, Rotate, and Translate

Use these members to apply a transform to a matrix.

rotateBy Applies a rotation transform to the matrix. A rotation transform rotates graphic objects. The last argument specifies the point around which the rotation occurs.

ITransformMatrix

```
ITransformMatrix&  
    rotateBy( double angle,  
              const IPoint& point = IPoint ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
IAccessError	Could not perform the rotate operation. Make sure you not attempting to access the object concurrently from another thread.	

scaleBy

Applies a scaling transform to the matrix. A scaling transform reduces or increases the size of graphic objects. The last argument specifies the point around which the scale occurs.

```
ITransformMatrix&  
    scaleBy( double xScale,  
            double yScale,  
            const IPoint& point = IPoint ( 0 , 0 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions		
IAccessError	Could not perform the scale operation. Make sure you not attempting to access the object concurrently from another thread.	

translateBy

Applies a translation transform to the matrix. A translation transform moves graphic objects.

```
ITransformMatrix&  
    translateBy( const IPoint& point );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

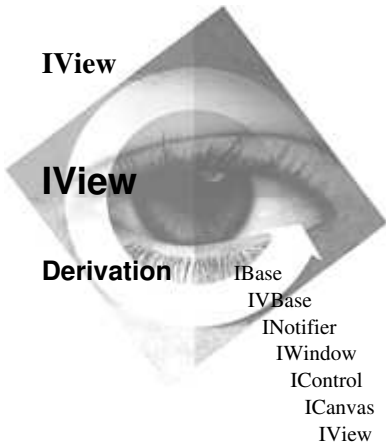
Exceptions		
IAccessError	Could not perform the translate operation. Make sure you not attempting to access the object concurrently from another thread.	

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IView

IView

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ICanvas
IView

Inherited By None.

Header File iview.hpp

Members				
	Member	Page	Member	Page
	Constructor	727	isDragSource	723
	bundle	722	isDropTarget	723
	component	723	model	723
	draw	725	moveSizeTo	724
	drawContents	725	setViewClient	724
	finalize	723	sizeTo	724
	handleNotification	726	viewClient	725
	initialize	724	~IView	726

Objects of the IView class control the display of data contained in the model and user interactions with the data.

The IView class cannot be constructed directly; you must create your own view class which derives from IView.

The derived class you create is one of the template parameters (along with the model) used to create your component stationery.

See IEmbedderModel (p. 243) for information about container embedder models, IModel (p. 703) for information about server models.

Public Functions

Informational

Use these members to obtain information about the view, and indicate drag and drop capabilities.

bundle Returns a reference to the component's bundle.

IView

IGUIBundle& bundle() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

component Returns a reference to the component.

IComponent& component() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

isDragSource

Indicates to the framework whether or not you want the view to be an OLE drag source. Returns true, unless overridden in your derived class.

Called by the framework for container components (those with models derived from IEmbedderModel). Has no other effects.

virtual Boolean isDragSource() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

isDropTarget Indicates to the framework whether or not you want the view to be registered as an OLE drop target. Returns true, unless overridden in your derived class.

Called by the framework for container components (those with models derived from IEmbedderModel). Has no other effects.

virtual Boolean isDropTarget() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

model Returns a pointer to the model. You should never cache this pointer, because it is subject to change: the model is deleted and recreated whenever the end-user selects options such as **File ► New** or **File ► Open**.

IModel* model() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

Initialization/Finalization

Use these members to initialize and close down views.

finalize Closes down the view. It does essentially the opposite of the initialize function.

If you call the initialize function of the base class you **must** call this function in order to shut down properly.

IView

<code>virtual void finalize();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

initialize Called by the framework to initialize the view. The default implementation starts the message handlers for painting, mouse events, and change notifications. If desired, it also registers the view as an OLE drop target (see `IView::isDropTarget`).

If you override this function to provide customized initialization for your view, it is recommended that you call the base class implementation.

<code>virtual void initialize();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Moving and Sizing

Use these members to move and size a view.

moveSizeTo Overrides the base class `moveSizeTo` function to also resize the associated view client (see `setViewClient`), if one exists.

<code>virtual IView& moveSizeTo(const IRectangle& newRect);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

setViewClient Allows you to attach a control (typically a multi-cell canvas) as the client of the view. The framework automatically sizes this control to the size of the view, so that the control always occupies the entire client area of the view (see the `sizeTo` and `moveSizeTo` functions). This function is typically used for applications that are servers and not for containers. In applications that are containers, embedded objects are obscured by the client control (which takes up the entire client area).

You can remove the attached control at any time by calling this method with `NULL`.

Note: The client control is NOT adopted by the view; you are responsible for freeing the memory when you are done.

<code>IView& setViewClient(IControl* clientControl);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

sizeTo Overrides the base class `sizeTo` function to also resize the associated view client (see `setViewClient`), if one exists.

<code>virtual IView& sizeTo(const ISize& newSize);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

IView

viewClient Returns a pointer to the view client control, or NULL if no client control is present.

<code>IControl*</code> <code>viewClient();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

Notification and Painting Event Handlers

Use these members to draw the contents of a view, and provide notification when the content changes.

draw Called by the framework whenever a “contents” presentation of the view is required. This is primarily in response to paint messages, but is also used to provide the presentations used in OLE caches (metafiles).

This function first calls out to `IView::drawContents` to allow the “native” data to be drawn. Then, if the component is a container, it iterates through all embedded objects and calls on them to draw.

<code>virtual void</code> <code>draw(IPresSpaceHandle&,</code> <code> const IRectangle&,</code> <code> Boolean metaFile = false);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

The parameters for this function are as follows:

IPresSpaceHandle&

The presentation space handle.

IRectangle&

Defines the rectangle that should be drawn.

Boolean Indicates that a metafile representation should be drawn for the embedded component.

drawContents

Called by the `IView::draw` function to draw the “native” data in the view. The default implementation does nothing, so you must override this function in your derived class in order to have the data show on the screen.

The framework provides a presentation space handle (`IPresSpaceHandle`) into which to draw, an invalid rectangle (`IRectangle`) that can be used to optimize drawing, and a `Boolean` which will be true if drawing to a metafile, or false if drawing to the screen. The metafile provides the cached presentation that OLE containers use to show the view while the server component is not running, or while it is running out of place.

IView

```
virtual void
drawContents( IPresSpaceHandle&,
              const IRectangle&,
              Boolean );
```

Win	PM	Motif
<i>Y</i>	<i>N</i>	<i>N</i>

The parameters for this function are as follows:

- IPresSpaceHandle&*
The presentation space handle.
- IRectangle&*
Defines the rectangle that should be drawn.
- Boolean*
Indicates that a metafile representation should be drawn for the embedded component.

handleNotification

Called by the framework in response to a change in the model. The base class implementation simply invalidates and repaints the entire view.

```
virtual void
handleNotification( const INotificationEvent& );
```

Win	PM	Motif
<i>Y</i>	<i>N</i>	<i>N</i>

The parameter for this function is as follows:

- INotificationEvent&*
A notification event that provides the details of a notification to an observer. Consult INotificationEvent (Vol. I) for more information about notification events.

Constructors

You cannot construct, copy, or destruct objects of this class directly.

~IView

```
virtual
~IView();
```

Win	PM	Motif
<i>Y</i>	<i>N</i>	<i>N</i>

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IView

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Constructors

You cannot construct, copy, or destruct objects of this class directly.

IView

1 IView(const IView&);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	N	N

Copying is not allowed.

2 IView(IGUIBundle&);

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	N	N

This is the default constructor for views. Views are constructed by the framework factory function IComponentStationery::createView (p. 16). Your view subclass must provide a constructor with the same parameter (IGUIBundle&), and must call this default constructor.

Inherited Protected Functions

IView

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Inherited Public Data

ICanvas		
classDefaultStyle		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Appendix A. Classes and Header Files Cross-Reference Tables

This appendix contains the following cross-reference tables:

“Classes to Header Files”

List of classes, showing the header file where each class can be found.

“Header Files to Classes” on page 748

List of header files, showing the classes that each header file contains.

Classes to Header Files

The following table shows the header file where a specific class can be found.

Class Name	Header File
Bag	ibag.h
Collection	iacllct.h
Constant Iterator	iiter.h
Cursor	icursor.h
Deque	ideque.h
Equality Collection	iaequal.h
Equality Key Collection	iaekey.h
Equality Key Sorted Collection	iaeqsrt.h
Equality Sequence	ieqseq.h
Equality Sorted Collection	iaeqsrt.h
Heap	iheap.h
I0String	i0string.hpp
I3StateCheckBox	i3statbx.hpp
I3StateCheckBox::Style	i3statbx.hpp
IAccelTblHandle	ihandle.hpp
IAccelerator	iaccel.hpp
IAcceleratorKey	iaccelky.hpp
IAcceleratorTable	iacceltb.hpp
IAcceleratorTable::Cursor	iacceltb.hpp
IAccessError	iexcbase.hpp

Classes and Header Files

Class Name	Header File
IAnchorBlockHandle	ihandle.hpp
IAnimatedButton	ianimbut.hpp
IAnimatedButton::Style	ianimbut.hpp
IApplication	iapp.hpp
IAssertionFailure	iexcbase.hpp
IAutoElemPointer	iptr.h
IAutoPointer	iptr.h
IBase	ibase.hpp
IBase::Version	ibase.hpp
IBaseComboBox	icombobs.hpp
IBaseComboBox::Cursor	icombobs.hpp
IBaseComboBox::Style	icombobs.hpp
IBaseErrorInfo	iexcept.hpp
IBaseListBox	ilistbas.hpp
IBaseListBox::Cursor	ilistbas.hpp
IBaseListBox::Style	ilistbas.hpp
IBaseSpinButton	ispinbas.hpp
IBaseSpinButton::Style	ispinbas.hpp
IBaseStream	ibasstrm.hpp
IBidiSettings	ibidiset.hpp
IBitFlag	ibitflag.hpp
IBitmapControl	ibmpctl.hpp
IBitmapControl::Style	ibmpctl.hpp
IBitmapHandle	ihandle.hpp
IBuffer	ibuffer.hpp
IButton	ibutton.hpp
IButton::Style	ibutton.hpp
IButtonNotifyHandler	ibtnnhdr.hpp
ICLibErrorInfo	iexcept.hpp
ICanvas	icanvas.hpp

Classes and Header Files

Class Name	Header File
ICanvas::Style	icanvas.hpp
ICheckBox	icheckbx.hpp
ICheckBox::Style	icheckbx.hpp
ICircularSlider	icslider.hpp
ICircularSlider::Style	icslider.hpp
ICircularSliderNotifyHandler	icsliden.hpp
IClipboard	iclipbrd.hpp
IClipboard::Cursor	iclipbrd.hpp
IClipboardHandler	icliphdr.hpp
ICnrAllocator	icnrobj.hpp
ICnrBeginEditEvent	icnreevt.hpp
ICnrControlList	icnrcfst.hpp
ICnrDate	idate.hpp
ICnrDrawBackgroundEvent	icnrdiev.hpp
ICnrDrawHandler	icnrhdh.hpp
ICnrDrawItemEvent	icnrdiev.hpp
ICnrEditEvent	icnreevt.hpp
ICnrEditHandler	icnrhdr.hpp
ICnrEmphasisEvent	icnrevt.hpp
ICnrEndEditEvent	icnreevt.hpp
ICnrEnterEvent	icnrevt.hpp
ICnrEvent	icnrevt.hpp
ICnrHandler	icnrhdr.hpp
ICnrHelpEvent	icnrevt.hpp
ICnrMenuHandler	icnrmhdr.hpp
ICnrObjectSet	icnrolst.hpp
ICnrQueryDeltaEvent	icnrevt.hpp
ICnrReallocStringEvent	icnreevt.hpp
ICnrScrollEvent	icnrevt.hpp
ICnrTime	itime.hpp

Classes and Header Files

Class Name	Header File
ICollectionViewComboBox	icombovw.hpp
ICollectionViewConstants	icollvwi.hpp
ICollectionViewListBox	ilistcvw.hpp
IColor	icolor.hpp
IComboBox	icombobx.hpp
IComboBox::Style	icombobx.hpp
IComboBoxNotifyHandler	icombonh.hpp
ICommand	icmd.hpp
ICommandConnectionTo	icmdhdr.hpp
ICommandEvent	icmdevt.hpp
ICommandHandler	icmdhdr.hpp
IComponent	icompnen.hpp
IComponentStationery	istatnry.hpp
IComponentStationeryFor	istatnry.hpp
IContainerColumn	icnrcol.hpp
IContainerControl	icnrctl.hpp
IContainerControl::Attribute	icnrctl.hpp
IContainerControl::ColumnCursor	icnrctl.hpp
IContainerControl::CompareFn	icnrctl.hpp
IContainerControl::FilterFn	icnrctl.hpp
IContainerControl::Iterator	icnrctl.hpp
IContainerControl::ObjectCursor	icnrctl.hpp
IContainerControl::Style	icnrctl.hpp
IContainerControl::TextCursor	icnrctl.hpp
IContainerControlNotifyHandler	icnrnhdr.hpp
IContainerObject	icnrobj.hpp
IContext	icontext.hpp
IContextHandle	ihandle.hpp
IControl	icontrol.hpp
IControl::Style	icontrol.hpp

Classes and Header Files

Class Name	Header File
IControlEvent	ictlevt.hpp
ICoordinateSystem	icoordsy.hpp
ICritSec	icritsec.hpp
ICurrentApplication	iapp.hpp
ICurrentThread	ithread.hpp
ICustomButton	icustbut.hpp
ICustomButton::Style	icustbut.hpp
ICustomButtonDrawEvent	icustbev.hpp
ICustomButtonDrawHandler	icustbhd.hpp
IDBCSBuffer	idbcsbuf.hpp
IDDEAcknowledgeEvent	iddeevt.hpp
IDDEAcknowledgeExecuteEvent	iddeevt.hpp
IDDEAcknowledgePokeEvent	iddeevt.hpp
IDDEActiveServer	iddecset.hpp
IDDEActiveServerSet	iddecset.hpp
IDDEBeginEvent	iddeevt.hpp
IDDEClientAcknowledgeEvent	iddeevt.hpp
IDDEClientConversation	iddeccnv.hpp
IDDEClientEndEvent	iddeevt.hpp
IDDEClientHotLinkEvent	iddeevt.hpp
IDDEClientHotLinkSet	iddecset.hpp
IDDEDataEvent	iddeevt.hpp
IDDEEndEvent	iddeevt.hpp
IDDEEvent	iddeevt.hpp
IDDEExecuteEvent	iddeevt.hpp
IDDEPokeEvent	iddeevt.hpp
IDDERequestDataEvent	iddeevt.hpp
IDDEServerAcknowledgeEvent	iddeevt.hpp
IDDEServerHotLinkEvent	iddeevt.hpp
IDDESetAcknowledgeInfoEvent	iddeevt.hpp

Classes and Header Files

Class Name	Header File
IDDETopicServer	iddetsrv.hpp
IDM	idmcomm.hpp
IDMCnrItem	idmcnrit.hpp
IDMEFItem	idmefit.hpp
IDMEvent	idmevent.hpp
IDMHandler	idmhndlr.hpp
IDMImage	idmimage.hpp
IDMImage::Style	idmimage.hpp
IDMItem	idmitem.hpp
IDMItemProvider	idmprov.hpp
IDMItemProviderFor	idmprov.hpp
IDMMLEItem	idmmleit.hpp
IDMMenuItem	idmmenit.hpp
IDMOperation	idmoper.hpp
IDMRenderer	idmrendr.hpp
IDMSourceBeginEvent	idmevent.hpp
IDMSourceDiscardEvent	idmevent.hpp
IDMSourceEndEvent	idmevent.hpp
IDMSourceHandler	idmsrch.hpp
IDMSourceOperation	idmsrcop.hpp
IDMSourcePrepareEvent	idmevent.hpp
IDMSourcePrintEvent	idmevent.hpp
IDMSourceRenderEvent	idmevent.hpp
IDMSourceRenderer	idmsrcrn.hpp
IDMTBarButtonItem	idmtbbit.hpp
IDMTargetDropEvent	idmevent.hpp
IDMTargetEndEvent	idmevent.hpp
IDMTargetEnterEvent	idmevent.hpp
IDMTargetEvent	idmevent.hpp
IDMTargetHandler	idmtgth.hpp

Classes and Header Files

Class Name	Header File
IDMTargetHelpEvent	idmevent.hpp
IDMTargetLeaveEvent	idmevent.hpp
IDMTargetOperation	idmtgtop.hpp
IDMTargetRenderer	idmtgtrn.hpp
IDMToolBarItem	idmtbrit.hpp
IDate	idate.hpp
IDeviceColor	icolor.hpp
IDeviceError	iexcbase.hpp
IDisplayHandle	ihandle.hpp
IDocumentStorage	idocstor.hpp
IDrawItemEvent	idievt.hpp
IDrawingCanvas	idrawcv.hpp
IDrawingCanvas::Style	idrawcv.hpp
IDynamicLinkLibrary	ireslib.hpp
IEditHandler	iedithdr.hpp
IElemPointer	iptr.h
IEmbeddedComponent	iembmod.hpp
IEmbedderModel	iembmod.hpp
IEntryField	ientryfd.hpp
IEntryField::Style	ientryfd.hpp
IEntryFieldNotifyHandler	ientrynh.hpp
IEnumHandle	ihandle.hpp
IEvent	ievent.hpp
IEventData	ievtdat.hpp
IEventParameter1	ievtdat2.hpp
IEventParameter2	ievtdat2.hpp
IEventResult	ievtdat2.hpp
IException	iexcbase.hpp
IException::TraceFn	iexcbase.hpp
IExceptionLocation	iexcbase.hpp

Classes and Header Files

Class Name	Header File
FileDialog	ifiledlg.hpp
FileDialog::Settings	ifiledlg.hpp
FileDialog::Style	ifiledlg.hpp
FileDialogEvent	ifilehdr.hpp
FileDialogHandler	ifilehdr.hpp
FileStream	ifilstrm.hpp
IFlatFileStorage	idocstor.hpp
IFlyOverHelpHandler	iflyhhdr.hpp
IFlyText	iflytext.hpp
IFocusHandler	ifocshdr.hpp
IFont	ifont.hpp
IFont::FaceNameCursor	ifont.hpp
IFont::PointSizeCursor	ifont.hpp
IFontDialog	ifontdlg.hpp
IFontDialog::Settings	ifontdlg.hpp
IFontDialog::Style	ifontdlg.hpp
IFontDialogHandler	ifonthdr.hpp
IFrameEvent	iframevt.hpp
IFrameExtension	iframeext.hpp
IFrameExtensions	iframeext.hpp
IFrameFormatEvent	iframevt.hpp
IFrameHandler	iframhdr.hpp
IFrameWindow	iframe.hpp
IFrameWindow::Style	iframe.hpp
IFrameWindowNotifyHandler	iframnhd.hpp
IG3PointArc	igarc.hpp
IGArc	igarc.hpp
IGBitmap	igbitmap.hpp
IGChord	igpie.hpp
IGEllipse	igelipse.hpp

Classes and Header Files

Class Name	Header File
IGLine	igline.hpp
IGList	iglist.hpp
IGList::Cursor	iglist.hpp
IGPie	igpie.hpp
IGPolygon	igpyline.hpp
IGPolyline	igpyline.hpp
IGRectangle	igrect.hpp
IGRegion	igregion.hpp
IGString	igstring.hpp
IGUIBundle	iguibndl.hpp
IGUIColor	icolor.hpp
IGUIErrorInfo	iexcept.hpp
IGrabHandles	igrabhdl.hpp
IGraphic	igraphic.hpp
IGraphicBundle	igbundle.hpp
IGraphicContext	igrafctx.hpp
IGraphicPushButton	igraphbt.hpp
IGraphicPushButton::Style	igraphbt.hpp
IGroupBox	igroupbx.hpp
IGroupBox::Style	igroupbx.hpp
IHandle	ibhandle.hpp
IHandler	ihandler.hpp
IHelpErrorEvent	ihelpvt.hpp
IHelpHandler	ihelphdr.hpp
IHelpHyperlinkEvent	ihelpvt.hpp
IHelpMenuBarEvent	ihelpvt.hpp
IHelpNotifyEvent	ihelpvt.hpp
IHelpSubitemNotFoundEvent	ihelpvt.hpp
IHelpTutorialEvent	ihelpvt.hpp
IHelpWindow	ihelp.hpp

Classes and Header Files

Class Name	Header File
IHelpWindow::Settings	ihelp.hpp
IHelpWindow::Style	ihelp.hpp
IHighEventParameter	ievtdat2.hpp
IIconControl	iiconctl.hpp
IIconControl::Style	iiconctl.hpp
IInfoArea	iinfoa.hpp
IInvalidParameter	iexcbase.hpp
IInvalidRequest	iexcbase.hpp
IKey	key.hpp
IKey::KeyModifier	key.hpp
IKeyboardConnectionTo	keyhdr.hpp
IKeyboardEvent	keyevt.hpp
IKeyboardHandler	keyhdr.hpp
IListBox	listbox.hpp
IListBox::Style	listbox.hpp
IListBoxDrawItemEvent	ilbdielt.hpp
IListBoxDrawItemHandler	ilbdihdr.hpp
IListBoxNotifyHandler	listbnh.hpp
IListBoxSizeItemEvent	ilbdielt.hpp
ILowEventParameter	ievtdat2.hpp
IMGrabbable	grabhdl.hpp
IMM24FramesPerSecondTime	immtime.hpp
IMM25FramesPerSecondTime	immtime.hpp
IMM30FramesPerSecondTime	immtime.hpp
IMMAmpMixer	immamix.hpp
IMMAudioBuffer	immabuf.hpp
IMMAudioCD	immcdla.hpp
IMMAudioCDContents	immcdla.hpp
IMMAudioCDContents::Cursor	immcdla.hpp
IMMCDXA	immcdxa.hpp

Classes and Header Files

Class Name	Header File
IMMConfigurableAudio	immaud.hpp
IMMCuePointEvent	immevt.hpp
IMMDevice	immdev.hpp
IMMDeviceEvent	immevt.hpp
IMMDeviceHandler	immdevhdr.hpp
IMMDeviceNotifyHandler	immdevnh.hpp
IMMDigitalVideo	immdivd.hpp
IMMErrorInfo	immexcpt.hpp
IMMFileMedia	immfilem.hpp
IMMHourMinSecFrameTime	immtime.hpp
IMMHourMinSecTime	immtime.hpp
IMMMasterAudio	immmaud.hpp
IMMMillisecondTime	immtime.hpp
IMMMinSecFrameTime	immtime.hpp
IMMNotifyEvent	immevt.hpp
IMMPassDeviceEvent	immevt.hpp
IMMPlayableDevice	immplayd.hpp
IMMPlayerPanel	immplypn.hpp
IMMPlayerPanelHandler	immplyhd.hpp
IMMPositionChangeEvent	immevt.hpp
IMMRecordable	immrecrd.hpp
IMMRemovableMedia	immremmed.hpp
IMMRemovableMediaHandler	immremhd.hpp
IMMRemovableMediaNotifyHandler	immremnh.hpp
IMMSequencer	immsequ.hpp
IMMSpeed	immspeed.hpp
IMMTime	immtime.hpp
IMMTrackMinSecFrameTime	immtime.hpp
IMMWaveAudio	immwave.hpp
IMMemoryStream	imemstrm.hpp

Classes and Header Files

Class Name	Header File
IMenu	imenu.hpp
IMenu::Cursor	imenu.hpp
IMenu::Style	imenu.hpp
IMenuBar	imenubar.hpp
IMenuBar::Style	imenubar.hpp
IMenuDrawItemEvent	imndievt.hpp
IMenuDrawItemHandler	imndihdr.hpp
IMenuDrawItemHandler::DrawFlag	imndihdr.hpp
IMenuEvent	imenuevt.hpp
IMenuHandle	ihandle.hpp
IMenuHandler	imenuhdr.hpp
IMenuItem	imnitem.hpp
IMenuItem::Attribute	imnitem.hpp
IMenuItem::Style	imnitem.hpp
IMenuNotifyHandler	imenunhd.hpp
IMessageBox	imsgbox.hpp
IMessageBox::Style	imsgbox.hpp
IMessageQueueHandle	ihandle.hpp
IMessageText	imsgtext.hpp
IMetaType	imetatyp.hpp
IMetaTypeInfo	imetatyp.hpp
IMngElemPointer	iptr.h
IMngPointer	iptr.h
IModel	imodel.hpp
IModuleHandle	ihandle.hpp
IMouseClickEvent	imousevt.hpp
IMouseConnectionTo	imoushdr.hpp
IMouseEvent	imousevt.hpp
IMouseHandler	imoushdr.hpp
IMousePointerEvent	imousevt.hpp

Classes and Header Files

Class Name	Header File
IMousePointerHandler	imphdr.hpp
IMultiCellCanvas	imcelcv.hpp
IMultiCellCanvas::Style	imcelcv.hpp
IMultiLineEdit	imle.hpp
IMultiLineEdit::Style	imle.hpp
IMultiLineEditNotifyHandler	imlenhdr.hpp
INotebook	inotebk.hpp
INotebook::Cursor	inotebk.hpp
INotebook::PageSettings	inotebk.hpp
INotebook::PageSettings::Attribute	inotebk.hpp
INotebook::Style	inotebk.hpp
INotebookDrawItemEvent	inbdievt.hpp
INotebookNotifyHandler	inotebnh.hpp
INotificationEvent	inotifev.hpp
INotifier	inotify.hpp
INumericSpinButton	ispinnum.hpp
INumericSpinButton::Style	ispinnum.hpp
INumericSpinButtonNotifyHandler	ispbtnnh.hpp
IObjectWindow	iobjwin.hpp
IObserver	iobsrvr.hpp
IObserverList	iobslist.hpp
IObserverList::Cursor	iobslist.hpp
IOutOfMemory	iexcbase.hpp
IOutOfSystemResource	iexcbase.hpp
IOutOfWindowResource	iexcbase.hpp
IOutlineBox	ioutlbox.hpp
IOutlineBox::Style	ioutlbox.hpp
IPageEvent	ipageevt.hpp
IPageHandle	inotebk.hpp
IPageHandler	ipagehdr.hpp

Classes and Header Files

Class Name	Header File
IPageHelpEvent	ipageevt.hpp
IPageRemoveEvent	ipageevt.hpp
IPageSelectEvent	ipageevt.hpp
IPaintConnectionTo	ipainhdr.hpp
IPaintEvent	ipainevt.hpp
IPaintHandler	ipainhdr.hpp
IPair	ipoint.hpp
IPoint	ipoint.hpp
IPointArray	iptarray.hpp
IPointerHandle	ihandle.hpp
IPopUpMenu	ipopmenu.hpp
IPresSpaceHandle	ihandle.hpp
IPrivateResource	ireslock.hpp
IProcedureAddress	iprocadr.hpp
IProcessId	ihandle.hpp
IProfile	iprofile.hpp
IProfile::Cursor	iprofile.hpp
IProfileHandle	ihandle.hpp
IProgressIndicator	islider.hpp
IProgressIndicator::Style	islider.hpp
IProgressIndicatorNotifyHandler	islidenh.hpp
IPushButton	ipushbut.hpp
IPushButton::Style	ipushbut.hpp
IRadioButton	iradiobt.hpp
IRadioButton::Style	iradiobt.hpp
IRange	ipoint.hpp
IRecoordHandler	irecohdr.hpp
IRectangle	irect.hpp
IRefCounted	irefcnt.hpp
IReference	irefcnt.hpp

Classes and Header Files

Class Name	Header File
IRegionHandle	ihandle.hpp
IResizeEvent	isizeevt.hpp
IResizeHandler	isizehdr.hpp
IResource	ireslock.hpp
IResourceExhausted	iexcbase.hpp
IResourceId	ireslib.hpp
IResourceLibrary	ireslib.hpp
IResourceLock	ireslock.hpp
ISWP	iswp.hpp
ISWPArray	iswp.hpp
IScrollBar	isroll.hpp
IScrollBar::Style	isroll.hpp
IScrollBarNotifyHandler	isclnhdr.hpp
IScrollEvent	iscllevt.hpp
IScrollHandler	isclhdr.hpp
ISelectHandler	iselhdr.hpp
ISemaphoreHandle	ibhandle.hpp
ISetCanvas	isetcv.hpp
ISetCanvas::Style	isetcv.hpp
ISettingButton	isetbut.hpp
ISettingButtonNotifyHandler	isetbnhd.hpp
ISharedResource	ireslock.hpp
IShowListHandler	islhdr.hpp
ISize	ipoint.hpp
ISlider	islider.hpp
ISlider::Style	islider.hpp
ISliderArmHandler	islдахdr.hpp
ISliderDrawHandler	islidhdr.hpp
ISpinHandler	ispinhdr.hpp
ISplitCanvas	isplitev.hpp

Classes and Header Files

Class Name	Header File
ISplitCanvas::Style	isplitcv.hpp
IStandardNotifier	istdntfy.hpp
IStaticText	istattxt.hpp
IStaticText::Style	istattxt.hpp
IString	istring.hpp
IStringEnum	istrenum.hpp
IStringGenerator	istrngen.hpp
IStringGeneratorFn	istrngen.hpp
IStringGeneratorMemberFn	istrngen.hpp
IStringGeneratorRefMemberFn	istrngen.hpp
IStringHandle	ihandle.hpp
IStringParser	istparse.hpp
IStringParser::SkipWords	istparse.hpp
IStringTest	istrtest.hpp
IStringTestMemberFn	istrtest.hpp
IStructuredStorage	idocstor.hpp
ISubmenu	isubmenu.hpp
ISubmenu::Cursor	isubmenu.hpp
ISystemBitmapHandle	ihandle.hpp
ISystemErrorInfo	iexcept.hpp
ISystemMenu	isysmenu.hpp
ISystemPointerHandle	ihandle.hpp
ITextControl	itextctl.hpp
ITextControlNotifyHandler	itextcnh.hpp
ITextSpinButton	ispintxt.hpp
ITextSpinButton::Cursor	ispintxt.hpp
ITextSpinButton::Style	ispintxt.hpp
ITextSpinButtonNotifyHandler	ispbttnh.hpp
IThread	ithread.hpp
IThread::Cursor	ithread.hpp

Classes and Header Files

Class Name	Header File
IThreadFn	ithread.hpp
IThreadHandle	ihandle.hpp
IThreadId	ihandle.hpp
IThreadMemberFn	ithread.hpp
ITime	itime.hpp
ITimer	itimer.hpp
ITimer::Cursor	itimer.hpp
ITimerFn	itimer.hpp
ITimerMemberFn	itimer.hpp
ITimerMemberFn0	itimer.hpp
ITitle	ititle.hpp
ITitleNotifyHandler	ititlenh.hpp
IToolBar	itbar.hpp
IToolBar::FrameCursor	itbar.hpp
IToolBar::Style	itbar.hpp
IToolBar::WindowCursor	itbar.hpp
IToolBarButton	itbarbut.hpp
IToolBarButton::Style	itbarbut.hpp
IToolBarContainer	itbarcnr.hpp
IToolBarContainer::Style	itbarcnr.hpp
IToolBarFrameWindow	itbarfrm.hpp
ITrace	itrace.hpp
ITransformMatrix	itrnsfrm.hpp
IVBase	ivbase.hpp
IView	iview.hpp
IViewPort	ivport.hpp
IViewPort::Style	ivport.hpp
IWindow	iwindow.hpp
IWindow::ChildCursor	iwindow.hpp
IWindow::ExceptionFn	iwindow.hpp

Classes and Header Files

Class Name	Header File
IWindow::Style	iwindow.hpp
IWindowHandle	ihandle.hpp
IWindowNotifyHandler	iwinhdr.hpp
IXLibErrorInfo	iexcept.hpp
Iterator	iiter.h
Key Bag	keybag.h
Key Collection	iakey.h
Key Set	keyset.h
Key Sorted Bag	ksbag.h
Key Sorted Collection	ksrt.h
Key Sorted Set	ksset.h
Manipulators	iomanip.h
Map	imap.h
Ordered Collection	iaorder.h
Pointer Classes	iptr.h
Priority Queue	iprioqu.h
Queue	iqueue.h
Relation	irel.h
Sequence	iseq.h
Sequential Collection	iasqntl.h
Set	iset.h
Sorted Bag	isrtbag.h
Sorted Collection	iasrt.h
Sorted Map	isrtmap.h
Sorted Relation	isrtrel.h
Sorted Set	isrtset.h
Stack	istack.h
Tree	itree.h
Tree Cursor	itcursor.h
c_exception	complex.h

Classes and Header Files

Class Name	Header File
complex	complex.h
filebuf	fstream.h
fstream	fstream.h
fstreambase	fstream.h
ifstream	fstream.h
ios	iostream.h
iostream	iostream.h
iostream_withassign	iostream.h
istream	iostream.h
istream_withassign	iostream.h
istrstream	iostream.h
ofstream	fstream.h
ostream	iostream.h
ostream_withassign	iostream.h
ostrstream	iostream.h
stdiobuf	stdiostream.h
stdiostream	stdiostream.h
streambuf	iostream.h
strstream	iostream.h
strstreambase	iostream.h
strstreambuf	strstream.h

Classes and Header Files

Header Files to Classes

The following table shows the User Interface Class Library header files and the classes that they contain.

Header File	Classes
complex.h	c_exception complex
fstream.h	filebuf fstream fstreambase ifstream ofstream
iOString.hpp	I0String
i3statbx.hpp	I3StateCheckBox::Style I3StateCheckBox
iaccel.hpp	IAccelerator
iaccelky.hpp	IAcceleratorKey
iacceltb.hpp	IAcceleratorTable::Cursor IAcceleratorTable
iacllct.h	Collection
iaeqkey.h	Equality Key Collection
iaeqsrt.h	Equality Key Sorted Collection
iaeqsrt.h	Equality Sorted Collection
iaequal.h	Equality Collection
iakey.h	Key Collection
iaksrt.h	Key Sorted Collection
ianimbut.hpp	IAnimatedButton::Style IAnimatedButton
iaorder.h	Ordered Collection
iapp.hpp	IApplication ICurrentApplication
iasqntl.h	Sequential Collection
iasrt.h	Sorted Collection
ibag.h	Bag
ibase.hpp	IBase::Version IBase

Classes and Header Files

Header File	Classes
ibasstrm.hpp	IBaseStream
ibhandle.hpp	IHandle ISemaphoreHandle
ibidiset.hpp	IBidiSettings
ibitflag.hpp	IBitFlag
ibmpctl.hpp	IBitmapControl::Style IBitmapControl
ibtnnhdr.hpp	IButtonNotifyHandler
ibuffer.hpp	IBuffer
ibutton.hpp	IButton IButton::Style
icanvas.hpp	ICanvas::Style ICanvas
icheckbx.hpp	ICheckBox::Style ICheckBox
iclipbrd.hpp	IClipboard::Cursor IClipboard
icliphdr.hpp	IClipboardHandler
icmd.hpp	ICommand
icmdevt.hpp	ICommandEvent
icmdhdr.hpp	ICommandHandler ICommandConnectionTo
icnrcfst.hpp	ICnrControlList
icnrcol.hpp	IContainerColumn
icnrctl.hpp	IContainerControl::ObjectCursor IContainerControl::Style IContainerControl::TextCursor IContainerControl IContainerControl::Attribute IContainerControl::ColumnCursor IContainerControl::CompareFn IContainerControl::FilterFn IContainerControl::Iterator
icnrdiv.hpp	ICnrDrawItemEvent ICnrDrawBackgroundEvent
icnrdivd.hpp	ICnrDrawHandler

Classes and Header Files

Header File	Classes
icnreevt.hpp	ICnrEditEvent ICnrEndEditEvent ICnrReallocStringEvent ICnrBeginEditEvent
icnrehdr.hpp	ICnrEditHandler
icnrevt.hpp	ICnrScrollEvent ICnrEnterEvent ICnrEvent ICnrHelpEvent ICnrQueryDeltaEvent ICnrEmphasisEvent
icnrhdr.hpp	ICnrHandler
icnrmhdr.hpp	ICnrMenuHandler
icnrnhdr.hpp	IContainerControlNotifyHandler
icnrobj.hpp	IContainerObject ICnrAllocator
icnrolst.hpp	ICnrObjectSet
icollvwi.hpp	ICollectionViewConstants
icolor.hpp	IGUIColor IColor IDeviceColor
icombobs.hpp	IBaseComboBox::Cursor IBaseComboBox::Style IBaseComboBox
icombobx.hpp	IComboBox::Style IComboBox
icombonh.hpp	IComboBoxNotifyHandler
icombovw.hpp	ICollectionViewComboBox
icompnen.hpp	IComponent
icontext.hpp	IContext
icontrol.hpp	IControl::Style IControl
icoordsy.hpp	ICoordinateSystem
icritsec.hpp	ICritSec
icsliden.hpp	ICircularSliderNotifyHandler

Classes and Header Files

Header File	Classes
icslider.hpp	ICircularSlider ICircularSlider::Style
ictlevt.hpp	IControlEvent
icursor.h	Cursor
icustbev.hpp	ICustomButtonDrawEvent
icustbhd.hpp	ICustomButtonDrawHandler
icustbut.hpp	ICustomButton ICustomButton::Style
idate.hpp	IDate
idbcsbuf.hpp	IDBCSBuffer
iddeccnv.hpp	IDDEClientConversation
iddecset.hpp	IDDEClientHotLinkSet IDDEActiveServerSet IDDEActiveServer
iddeevt.hpp	IDDEBeginEvent IDDEClientAcknowledgeEvent IDDEAcknowledgePokeEvent IDDEClientEndEvent IDDEClientHotLinkEvent IDDEAcknowledgeEvent IDDEDataEvent IDDEEndEvent IDDEEvent IDDEExecuteEvent IDDEPokeEvent IDDERequestDataEvent IDDEServerAcknowledgeEvent IDDEServerHotLinkEvent IDDESetAcknowledgeInfoEvent IDDEAcknowledgeExecuteEvent
iddetsrv.hpp	IDDETopicServer
ideque.h	Deque
idievt.hpp	IDrawItemEvent
idmcnrit.hpp	IDMCnrItem
idmcomm.hpp	IDM
idmefit.hpp	IDMEFItem

Classes and Header Files

Header File	Classes
idmevent.hpp	IDMSourcePrintEvent IDMSourceRenderEvent IDMTargetDropEvent IDMTargetEndEvent IDMTargetEnterEvent IDMTargetEvent IDMTargetHelpEvent IDMTargetLeaveEvent IDMEvent IDMSourceBeginEvent IDMSourceDiscardEvent IDMSourceEndEvent IDMSourcePrepareEvent
idmhndlr.hpp	IDMHandler
idmimage.hpp	IDMImage IDMImage::Style
idmitem.hpp	IDMItem
idmmenit.hpp	IDMMenuItem
idmmleit.hpp	IDMMLEItem
idmoper.hpp	IDMOperation
idmprov.hpp	IDMItemProviderFor IDMItemProvider
idmrendr.hpp	IDMRenderer
idmsrch.hpp	IDMSourceHandler
idmsrcop.hpp	IDMSourceOperation
idmsrcrn.hpp	IDMSourceRenderer
idmtbbit.hpp	IDMTBarButtonItem
idmtbrit.hpp	IDMToolBarItem
idmtgth.hpp	IDMTargetHandler
idmtgtop.hpp	IDMTargetOperation
idmtgrn.hpp	IDMTargetRenderer
idocstor.hpp	IFlatFileStorage IDocumentStorage IStructuredStorage
idrawcv.hpp	IDrawingCanvas::Style IDrawingCanvas

Classes and Header Files

Header File	Classes
iedithdr.hpp	IEditHandler
iembmod.hpp	IEmbedderModel IEmbeddedComponent
ientryfd.hpp	IEntryField::Style IEntryField
ientrynh.hpp	IEntryFieldNotifyHandler
ieqseq.h	Equality Sequence
ievent.hpp	IEvent
ievtat2.hpp	ILowEventParameter IEventParameter1 IEventParameter2 IEventResult IHighEventParameter
ievtdata.hpp	IEventData
iexcbase.hpp	IAccessError IDeviceError IException IInvalidParameter IInvalidRequest IException::TraceFn IOutOfMemory IOutOfSystemResource IOutOfWindowResource IResourceExhausted IExceptionLocation IAssertionFailure
ieexcept.hpp	ICLibErrorInfo ISystemErrorInfo IXLibErrorInfo IGUIErrorInfo IBaseErrorInfo
ifiledlg.hpp	IFileDialog::Settings IFileDialog::Style IFileDialog
ifilehdr.hpp	IFileDialogEvent IFileDialogHandler
ifilstrm.hpp	IFileStream
iflyhdr.hpp	IFlyOverHelpHandler
iflytext.hpp	IFlyText

Classes and Header Files

Header File	Classes
ifocshdr.hpp	IFocusHandler
ifont.hpp	IFont IFont::FaceNameCursor IFont::PointSizeCursor
ifontdlg.hpp	IFontDialog IFontDialog::Settings IFontDialog::Style
ifonthdr.hpp	IFontDialogHandler
iframe.hpp	IFrameWindow::Style IFrameWindow
iframevt.hpp	IFrameFormatEvent IFrameEvent
iframeext.hpp	IFrameExtension IFrameExtensions
iframhdr.hpp	IFrameHandler
iframnhd.hpp	IFrameWindowNotifyHandler
igarc.hpp	IGArc IG3PointArc
igbitmap.hpp	IGBitmap
igbundle.hpp	IGraphicBundle
igelipse.hpp	IGEllipse
igline.hpp	IGLine
iglist.hpp	IGList IGList::Cursor
igpie.hpp	IGChord IGPie
igpyline.hpp	IGPolygon IGPolyline
igrabhdl.hpp	IGrabHandles IMGrabbable
igrafctx.hpp	IGraphicContext
igraphbt.hpp	IGraphicPushButton IGraphicPushButton::Style
igraphic.hpp	IGraphic
igrect.hpp	IGRectangle

Classes and Header Files

Header File	Classes
igregion.hpp	IGRegion
igroupbx.hpp	IGroupBox IGroupBox::Style
igstring.hpp	IGString
iguibndl.hpp	IGUIBundle
ihandle.hpp	IAcceleratorHandle ISystemBitmapHandle IMenuHandle ISystemPointerHandle IThreadHandle IThreadId IWindowHandle IMessageQueueHandle IModuleHandle IDisplayHandle IEnumHandle IBitmapHandle IPointerHandle IPresSpaceHandle IProcessId IProfileHandle IRegionHandle IAnchorBlockHandle IContextHandle IStringHandle
ihandler.hpp	IHandler
iheap.h	Heap
ihelp.hpp	IHelpWindow IHelpWindow::Settings IHelpWindow::Style
ihelpvt.hpp	IHelpErrorEvent IHelpHyperlinkEvent IHelpMenuBarEvent IHelpNotifyEvent IHelpSubitemNotFoundEvent IHelpTutorialEvent
ihelphdr.hpp	IHelpHandler
iiconctl.hpp	IIconControl IIconControl::Style
iinfoa.hpp	IInfoArea

Classes and Header Files

Header File	Classes
iiter.h	Constant Iterator Iterator
key.h	IKey::KeyModifier IKey
keybag.h	Key Bag
keyevt.h	IKeyboardEvent
keyhdr.h	IKeyboardConnectionTo IKeyboardHandler
keyset.h	Key Set
ksbag.h	Key Sorted Bag
ksset.h	Key Sorted Set
ilbdiect.h	IListBoxSizeItemEvent IListBoxDrawItemEvent
ilbdihdr.h	IListBoxDrawItemHandler
ilistbas.h	IBaseListBox::Cursor IBaseListBox::Style IBaseListBox
ilistbnh.h	IListBoxNotifyHandler
ilistbox.h	IListBox::Style IListBox
ilistcvw.h	ICollectionViewListBox
imap.h	Map
imcelcv.h	IMultiCellCanvas::Style IMultiCellCanvas
imemstrm.h	IMemoryStream
imenu.h	IMenu IMenu::Cursor IMenu::Style
imenubar.h	IMenuBar IMenuBar::Style
imenuevt.h	IMenuEvent
imenuhdr.h	IMenuHandler
imenuunhd.h	IMenuNotifyHandler

Classes and Header Files

Header File	Classes
imetatyp.hpp	IMetaType IMetaTypeInfo
imle.hpp	IMultiLineEdit::Style IMultiLineEdit
imlenhdr.hpp	IMultiLineEditNotifyHandler
immabuf.hpp	IMMAudioBuffer
immamix.hpp	IMMAmpMixer
immaud.hpp	IMMConfigurableAudio
immcdca.hpp	IMMAudioCD IMMAudioCDContents IMMAudioCDContents::Cursor
immcdxa.hpp	IMMCDXA
immdev.hpp	IMMDevice
immdevnh.hpp	IMMDeviceNotifyHandler
immdigvd.hpp	IMMDigitalVideo
immdvhdr.hpp	IMMDeviceHandler
immevt.hpp	IMMCuePointEvent IMMNotifyEvent IMMPassDeviceEvent IMMPositionChangeEvent IMMDeviceEvent
immexcpt.hpp	IMMErrorInfo
immfilem.hpp	IMMFileMedia
immmaud.hpp	IMMMasterAudio
immplayd.hpp	IMMPlayableDevice
immplyhd.hpp	IMMPlayerPanelHandler
immplypn.hpp	IMMPlayerPanel
immrecrd.hpp	IMMRecordable
immremed.hpp	IMMRemovableMedia
immremhd.hpp	IMMRemovableMediaHandler
immremnh.hpp	IMMRemovableMediaNotifyHandler
immsequ.hpp	IMMSequencer
immspeed.hpp	IMMSpeed

Classes and Header Files

Header File	Classes
immtime.hpp	IMMMillisecondTime IMMMinSecFrameTime IMM25FramesPerSecondTime IMM30FramesPerSecondTime IMMHourMinSecFrameTime IMMHourMinSecTime IMM24FramesPerSecondTime IMMTime
immttime.hpp	IMMTrackMinSecFrameTime
immwave.hpp	IMMWaveAudio
imndievt.hpp	IMenuDrawItemEvent
imndihdr.hpp	IMenuDrawItemHandler IMenuDrawItemHandler::DrawFlag
imnitem.hpp	IMenuItem::Style IMenuItem IMenuItem::Attribute
imodel.hpp	IModel
imousevt.hpp	IMouseClickEvent IMouseEvent IMousePointerEvent
imoushdr.hpp	IMouseHandler IMouseConnectionTo
imphdr.hpp	IMousePointerHandler
imsgbox.hpp	IMessageBox::Style IMessageBox
imsgtext.hpp	IMessageText
inbdievt.hpp	INotebookDrawItemEvent
inotebk.hpp	INotebook::PageSettings::Attribute IPageHandle INotebook::Style INotebook INotebook::Cursor INotebook::PageSettings
inotebnh.hpp	INotebookNotifyHandler
inotifev.hpp	INotificationEvent
inotify.hpp	INotifier
iobjwin.hpp	IObjectWindow

Classes and Header Files

Header File	Classes
iobsrvr.hpp	IObserver
iobslist.hpp	IObservableList IObservableList::Cursor
iomanip.h	Manipulators
iostream.h	iostream iostream_withassign istream istream_withassign istrstream ios ostream ostream_withassign ostrstream streambuf strstream strstreambase
ioutlbox.hpp	IOutlineBox IOutlineBox::Style
ipageevt.hpp	IPageSelectEvent IPageEvent IPageHelpEvent IPageRemoveEvent
ipagehdr.hpp	IPageHandler
ipainevt.hpp	IPaintEvent
ipainhdr.hpp	IPaintHandler IPaintConnectionTo
ipoint.hpp	IPair IPoint IRange ISize
ipopmenu.hpp	IPopUpMenu
iprioqu.h	Priority Queue
iprocadr.hpp	IProcedureAddress
iprofile.hpp	IProfile::Cursor IProfile
iptarray.hpp	IPointArray

Classes and Header Files

Header File	Classes
iptr.h	IMngPointer Pointer Classes IElemPointer IAutoElemPointer IAutoPointer IMngElemPointer
ipushbut.hpp	IPushButton::Style IPushButton
iqueue.h	Queue
iradiobt.hpp	IRadioButton IRadioButton::Style
irect.hpp	IRectangle
irefcnt.hpp	IReference IRefCounted
irel.h	Relation
ireslib.hpp	IResourceId IResourceLibrary IDynamicLinkLibrary
ireslock.hpp	IResourceLock IResource ISharedResource IPrivateResource
isclnhdr.hpp	IScrollBarNotifyHandler
iscllevt.hpp	IScrollEvent
isclhdr.hpp	IScrollHandler
iscroll.hpp	IScrollBar IScrollBar::Style
iselhdr.hpp	ISelectHandler
iseq.h	Sequence
iset.h	Set
isetbnhd.hpp	ISettingButtonNotifyHandler
isetbut.hpp	ISettingButton
isetcv.hpp	ISetCanvas ISetCanvas::Style
isizeevt.hpp	IResizeEvent

Classes and Header Files

Header File	Classes
isizehdr.hpp	IResizeHandler
isldahdr.hpp	ISliderArmHandler
islhdr.hpp	IShowListHandler
islidenh.hpp	IProgressIndicatorNotifyHandler
islider.hpp	ISlider::Style IProgressIndicator::Style IProgressIndicator ISlider
islidhdr.hpp	ISliderDrawHandler
ispbttnh.hpp	INumericSpinButtonNotifyHandler
ispbttnh.hpp	ITextSpinButtonNotifyHandler
ispinbas.hpp	IBaseSpinButton IBaseSpinButton::Style
ispinhdr.hpp	ISpinHandler
ispinnum.hpp	INumericSpinButton INumericSpinButton::Style
ispintxt.hpp	ITextSpinButton ITextSpinButton::Cursor ITextSpinButton::Style
isplitcv.hpp	ISplitCanvas ISplitCanvas::Style
isrtbag.h	Sorted Bag
isrtmap.h	Sorted Map
isrtrel.h	Sorted Relation
isrtset.h	Sorted Set
istack.h	Stack
istatnry.hpp	IComponentStationeryFor IComponentStationery
istattxt.hpp	IStaticText::Style IStaticText
istdntfy.hpp	IStandardNotifier
istparse.hpp	IStringParser IStringParser::SkipWords
istrenum.hpp	IStringEnum

Classes and Header Files

Header File	Classes
istrngen.hpp	IStrngGeneratorFn IStrngGeneratorMemberFn IStrngGeneratorRefMemberFn IStrngGenerator
istring.hpp	IStrng
istrtest.hpp	IStrngTest IStrngTestMemberFn
isubmenu.hpp	ISubmenu::Cursor ISubmenu
iswp.hpp	ISWPArray ISWP
isysmenu.hpp	ISystemMenu
itbar.hpp	IToolBar IToolBar::FrameCursor IToolBar::Style IToolBar::WindowCursor
itbarbut.hpp	IToolBarButton IToolBarButton::Style
itbarcnr.hpp	IToolBarContainer IToolBarContainer::Style
itbarfrm.hpp	IToolBarFrameWindow
itcursor.h	Tree Cursor
itextcnh.hpp	ITextControlNotifyHandler
itextctl.hpp	ITextControl
ithread.hpp	ICurrentThread IThreadFn IThread IThread::Cursor IThreadMemberFn
itime.hpp	ITime
itimer.hpp	ITimer ITimer::Cursor ITimerFn ITimerMemberFn ITimerMemberFn0
ititle.hpp	ITitle
ititlenh.hpp	ITitleNotifyHandler

Classes and Header Files

Header File	Classes
itrace.hpp	ITrace
itree.h	Tree
itrnsfrm.hpp	ITransformMatrix
ivbase.hpp	IVBase
iview.hpp	IView
ivport.hpp	IViewPort::Style IViewPort
iwindow.hpp	IWindow::Style IWindow::ChildCursor IWindow IWindow::ExceptionFn
iwinnhdr.hpp	IWindowNotifyHandler
stdiostream.h	stdiostream stdiobuf
strstream.h	strstreambuf



Glossary

This glossary defines terms and abbreviations that are used in this book. If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*, New York:McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

A

abstract class. (1) A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class. See also base class. (2) A class that allows polymorphism.

abstract data type. A mathematical model that includes a structure for storing data and operations that can be performed on that data. Common abstract data types include sets, trees, and heaps.

abstraction (data). See data abstraction.

access. An attribute that determines whether or not a class member is accessible in an expression or declaration. It can be public, protected, or private.

access declaration. A declaration used to adjust access to members of a base class.

access function. A function that returns information about the elements of an object so that you can analyze various elements of a string.

access resolution. The process by which the accessibility of a particular class member is determined.

access specifier. One of the C++ keywords public, private, or protected.

ambiguous derivation. A derivation where the class is derived from two or more base classes that have members with the same name.

amplifier. A device that increases the strength of input signals. Also referred to as an amp.

amplifier-mixer. A combination amplifier and mixer that is used to control the characters of an audio signal from one or more audio sources. Also referred to as an amp-mixer.

animate. Make or design in such a way as to create apparently spontaneous, lifelike movement.

animation rate. The number of thousandths of a second that pass before the next bitmap is displayed for a button while it is animated.

anonymous union. A union that is declared within a structure or class and that does not have a name.

area. In computer graphics, a filled shape, such as a solid rectangle.

array. An aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting.

array implementation. (In Collection Class Library) Implementation of an abstract data type using an array. Also called a tabular implementation.

ASCII (American National Standard Code for Information Interchange). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

Note: IBM has defined an extension to ASCII code (characters 128-255).

audio. Pertaining to the portion of recorded information that can be heard.

audio attributes. The standard audio attributes are: mute, volume, balance, treble, and bass.

audio formats. The way the audio information is stored and interpreted.

audio track. (1) The audio (sound) portion of the program. (2) The physical location where the audio is placed beside the image. (A system with two sound tracks can have either

automatic storage •CD-XA

stereo sound or two independent sound tracks.) Synonymous with sound track.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage*.

automatic storage management. The process that automatically allocates and deallocates objects in order to use memory efficiently.

auto-reset event. In Windows, an event used to signal a single thread that an application has completed.

See also event.

auxiliary classes. Classes that support other classes. Auxilliary classes in the Collection Class Library include classes for cursors, pointers and iterators.

AVL tree. A balanced binary search tree that does not allow the height of two siblings to differ by more than one.

B

B*-tree (B star tree). A tree in which only the leaves contain whole elements. All other nodes contain keys.

background color. The color in which the background of a graphic primitive is drawn.

balance. (1) For audio, refers to the relative strength of the left and right channels. A balance level of 0 is left channel only. A balance level of 100 is right channel only (2) A state of equilibrium, usually between treble and bass.

base class. A class from which other classes are derived. A base class may itself be derived from another base class. See also abstract class.

based on. A relationship between two classes in which one class is implemented through the other. A new class is “based on” an existing class when the existing class is used to implement it.

bass. The lower half of the whole vocal or instrumental tonal range.

bit field. A member of a structure or union that contains a specified number of bits.

bit mask. A pattern of characters used to control the retention or elimination of portions of another patterns of characters.

bits-per-sample. The number of bits of audio data that is to represent each sample of each channel (right or left). This is the resolution of the audio data. CD quality needs to be 16 bits-per-sample.

boundary alignment. The position in main storage of a fixed-length field (such as byte or doubleword) on an integral boundary for that unit of information.

For the Class Library example, a word boundary is a storage address evenly divisible by two.

bounded collection. A collection that has an upper limit on the number of elements it can contain.

brightness. The level of luminosity of the video signal. A brightness level of 0 produces a maximally white signal. A brightness level of 100 produces a maximally black signal.

built-in. A function that the compiler automatically puts inline instead of generating a call to the function.

C

camcorder. A compact, hand-held video camera with integrated videotape recorder.

canvas. Canvases are windows with a layout algorithm that manage child windows. The canvas classes are a set of window classes which allow you to implement dialog-like windows (that is, a window with several child controls). These windows are used for showing views of objects as both pages in a notebook and as windows that gather information to run an action. The different canvases can manage the size and position of child windows, provide moveable split bars between windows, and support the ability to scroll a window.

The canvases include the base class, ICanvas, and its four derived classes: IMultiCellCanvas, ISetCanvas, ISplitCanvas, and IViewport.

cast. A notation used to express the conversion of one type to another.

catch block. A block associated with a try block that receives control when a C++ exception matching its argument is thrown.

CD. Compact disc

CD-ROM. Compact disc-read-only memory

CD-XA. Compact disc-extended architecture

channel mapping •Compound Document Framework

channel mapping. The translation of a MIDI channel number for a sending device to an appropriate channel for a receiving device.

character array. An array of type char.

child. A node that is subordinate to another node in a tree structure. Only the root node of a tree is not a child.

child class. See derived class.

child window. A window derived from another window and drawn relative to it.

circular slider control. A 360-degree knob-like control that simulates the buttons on a TV, a stereo, or video components. By rotating the slider arm, the user can set, display, or modify a value, such as the balance, bass, volume, or treble.

class. A user-defined type. Classes can be defined hierarchically, allowing one class to be an expansion of another, and classes can restrict access to their members.

class hierarchy. A tree-like structure showing relationships among classes. It places one abstract class at the top (a base class) and one or more layers of derived classes below it.

class library. A collection of classes.

class template. A blueprint describing how a set of related classes can be constructed.

client area window. An intermediate window between an IFrameWindow and its controls and other child windows.

client program. A program that uses a class. The program is said to be a client of the class.

CLSID. The globally unique identifier for an object. The system Registry uses the CLSID to distinguish all OLE objects available on a system. A CLSID contains 32 hex digits.

collection. (1) In a general sense, an implementation of an abstract data type for storing elements. (2) An abstract class without any ordering, element properties, or key properties. All abstract Collection Classes are derived from Collection.

Collection Classes. A set of classes that implement abstract data types for storing elements.

color palette. A set of all the colors that can be used in a displayed image.

common controls. In Windows, a DLL that includes the following: a header control (a window for displaying

multiple columns of data), list view (a way to display objects as icons with labels), progress bar, property sheet, status bar, tool bar, track bar (slider control), tree view (an outline-type list), and an up-down control (spin control).

compact disc (CD). (1) A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser. (2) A disc with information stored in the form of pits along a spiral track. The information is decoded by a compact-disc player and interpreted as digital audio data, which most computers can process.

compact disc-extended architecture (CD-EX). A storage format that accommodates interleaved storage of audio, video, and standard file system data.

compact disc-read-only memory (CD-ROM). (1) An optical storage medium (2) High-capacity, read-only memory in the form of an optically read compact disc.

Complex Mathematics library. A C++ class library that provides the facilities to manipulate complex numbers and perform standard mathematical operations on them.

component. The unit of exchange in terms of OLE compound documents; also referred to as a document component, Components created using the Compound Document Framework are either container or server components.

component stationery. The “glue” that holds all pieces of a Compound Document Framework application together, including the model, view, and frame window. The template subclass IComponentStationeryFor, which takes a model and view as arguments, does the work of instantiating a component stationery. Synonym for stationery.

composite. The combination of two or more film, video, or electronic images into a single frame or display.

compound document. A single centralized location for integrating arbitrary or unstructured data from different sources. The Compound Document Framework provides a mechanism for creating document components by providing a structure that you can easily extend to create servers or containers. The framework stores compound documents using the OLE structured storage specification (that is, docfiles).

Compound Document Framework. A starting point for creating a server or container document component that is OLE-enabled. Compound documents allow for the integration of arbitrary or unstructured data from different sources into one centralized location.

computer-controlled device •delete

computer-controlled device. An external video source device with frame-stepping capability, usually a videodisc player, whose output can be controlled by the multimedia subsystem.

concrete class. A class that implements an abstract data type but does not allow polymorphism.

const. (1) An attribute of a data object that declares that the object cannot be changed. (2) An attribute of a function that declares that the function will not modify data members of its class.

constructor. A special class member function that has the same name as the class and is used to construct and possibly initialize objects of its class type. A return type is not specified.

container. A holder for zero or more embedded components. Containers manage the compound document by maintaining a list of embedded components and storing the container and its embedded component's data when requested. Containers built with the Compound Document Framework are also servers and can therefore be embedded inside of other containers.

containment function. A function that determines whether a collection contains a given element.

control. A graphic object that represents operations or properties of other objects.

See also tree control.

copy constructor. A constructor used to make a copy of an object from another object of the same type.

critical section. (1) Code that must be executed by one thread while all other threads in the process are suspended. (2) In Windows, a synchronization object. A critical section is not a kernel object; that is, it is not managed by the low-level components of the operating system and is not manipulated using handles. (3) In Windows, a small section of code that requires exclusive access to some shared data before the code can execute. Critical threads synchronize threads only within a single process, and they allow only one thread at a time to gain access to a region of data.

See also mutex, semaphore, and event. Contrast with kernel object.

cursor. A reference to an element at a specific position in a data structure.

cursor iteration. The process of repeatedly moving the cursor to the next element in a collection until some condition is satisfied.

cursored emphasis. When the selection cursor is on a choice, that choice has cursored emphasis.

C/2. A version of the C language designed for the OS/2 environment.

D

daemon. A program that runs unattended to perform a service for other programs.

data abstraction. A data type with a private representation and a public set of operations. The C++ language uses the concept of classes to implement data abstraction.

DBCS (Double-Byte Character Set). See double-byte character set.

deck. A line of child windows in a set canvas that is direction-independent. A horizontal deck is equivalent to a row and a vertical deck is equivalent to a column.

declaration. Introduces a name to a program and specifies how the name is to be interpreted.

declare. To specify the interpretation that C++ gives to each identifier.

default argument. An argument that is declared with a default value in a function prototype or declaration. If a call to the function omits this argument, the default value is used. Arguments with default values must be the trailing arguments in a function prototype argument list.

default class. A class with preprogrammed definitions that can be used for simple implementations.

default constructor. A constructor that takes no arguments, or a constructor for which all the arguments have default values.

default implementation. One of several possible implementation variants offered as the default for a specific abstract data type.

default operation class. A class with preprogrammed definitions for all required element and key operations for a particular implementation.

degree. The number of children of a node.

delete. (1) A C++ keyword that identifies a free-storage deallocation operator. (2) A C++ operator used to destroy objects created by operator new.

deque •element equality

deque. A queue that can have elements added and removed at both ends. A double-ended queue.

dequeue. An operation that removes the first element of a queue.

derivation. (1) The creation of a new or derived class from an existing base class. (2) The relationship between a class and the classes above or below it in a class hierarchy.

derived class. A class that inherits from a base class. You can add new data members and member functions to the derived class. You can manipulate a derived class object as if it were a base class object. The derived class can override virtual functions of the base class.

Synonym for child class and subclass.

destructor. A special member function that has the same name as its class, preceded by a tilde (~), and that “cleans up” after an object of that class, for example, by freeing storage that was allocated when the object was created. A destructor has no arguments, and no return type is specified.

difference. Given two sets A and B, the difference (A-B) is the set of all elements contained in A but not in B.

digital audio. Audio data that has been converted to digital form.

digital video. Material that can be seen and that has been converted to digital form.

digital video device. A full-motion video device that can record or play files (or both) containing digitally stored video.

diluted array. An array in which elements are deleted by being flagged as deleted, rather than by actually removing them from the array and shifting later elements to the left.

diluted sequence. A sequence implemented using a diluted array.

direct manipulation. A user interface technique whereby the user initiates application functions by manipulating the objects, represented by icons, on the Presentation Manager (PM) or Workplace Shell desktop. The user typically initiates an action by:

1. Selecting an icon
2. Pressing and holding down a mouse button while “dragging” the icon over another object’s icon on the desktop
3. Releasing the mouse button to “drop” the icon over the target object.

Thus, this technique is also known as “drag and drop” manipulation.

direct manipulation. In Windows, a unit of data. The unit is often part of a task that is shared among users.

document component. The basic unit of data exchange in the Compound Document Framework. A document component can be either a server or a container. Document components are commonly referred to as either documents or components.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, you need hardware and supporting software that are DBCS-enabled to enter, display, and print DBCS characters.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and can be addressed as a unit. For the C Set++ for AIX compiler, a doubleword is 32 bits (4 bytes).

drag after. A target enter event that occurs in a container where its `orderedTargetEmphasis` or `mixedTargetEmphasis` attribute is set and the current view is name, text, or details.

drag item. A “proxy” for the object being manipulated.

drag over. A target enter event that occurs in a container where its `orderedTargetEmphasis` attribute is not set and the current view is icon or tree view.

drop offset. The location where the next container object that is dropped will be positioned (if the target operation’s drop style is not `IDM::dropPosition`). The position is based upon the last object that was dropped as an offset of that object relative to the drop style.

dynamic casting. An intelligent mechanism to obtain the correct pointer to a base class.

E

element. The component of an array, subrange, enumeration, or set.

element equality. A relation that determines whether two elements are equal.

element function •frame extension

element function. A function, called by a member function, that accesses the elements of a class.

embedded component. Data created by another application that is stored in a container. Unlike a linked object, an embedded component does not have its own file on disk. Instead, it is stored in the container's structured storage file.

encapsulation. The hiding of the internal representation of objects and implementation details from the client program.

enqueue. An operation that adds an element as the last element to a queue.

enumeration constant. An identifier that is defined in an enumeration and that has an associated constant integer value. You can use an enumeration constant anywhere an integer constant is allowed.

enumeration data type. A type that represents integers and a set of enumeration constants. Each enumeration constant has an associated integer value.

equality collection. (1) An abstract class with the property of element equality. (2) In general, any collection that has element equality.

equality key collection. An abstract class with the properties of element equality and key equality.

equality key sorted collection. An abstract class with the properties of element equality, key equality, and sorted elements.

equality sequence. A sequentially ordered flat collection with element equality.

equality sorted collection. An abstract class with the properties of element equality and sorted elements.

event. In Windows, a synchronization kernel object used to signal that an operation has completed. See also kernel object. Compare to critical section, mutex, semaphore, manual-reset event, and auto-reset event.

exception. (1) A user or system error detected by the system and passed to an operating system or user exception handler. (2) For C++, any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called "throwing the exception").

exception handler. (1) A function that is invoked when an exception is detected, and that either corrects the problem and returns execution to the program, or terminates the program.

(2) In C++, a catch block that catches a C++ exception when it is thrown from a function in a try block.

exception handling. A type of error handling that allows control and information to be passed to an exception handler when an exception occurs. Under the OS/2 operating system, exceptions are generated by the system and handled by user code. In C++, try, catch, and throw expressions are the constructs used to implement C++ exception handling.

external data definition. A definition appearing outside a function. The defined object is accessible to all functions that follow the definition and are located within the same source file as the definition.

eyecatcher. A recognizable sequence of bytes that determines which parameters were passed in which registers. This sequence is used for functions that have not been prototyped or have a variable number of parameters.

F

file descriptor. A small positive integer that the system uses instead of the file name to identify an open file.

file scope. A name declared outside all blocks and classes has file scope and can be used after the point of declaration in a source file.

filter. A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream.

first element. The element visited first in an iteration over a collection. Each collection has its own definition for first element. For example, the first element of a sorted set is the element with the smallest value.

flat collection. A collection that has no hierarchical structure.

folder. A directory.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

frame. (1) A complete television picture that is composed of two scanned fields, one of the even lines and one of the odd lines. In the NTSC system, a frame has 525 horizontal lines and is scanned in 1/30th of a second. (2) A border around a window.

frame extension. A control you can add if it is not available in the basic Presentation Manager frame windows.

frame number • initializer

frame number. (1) The number used to identify a frame. (2) The location of a frame on a videodisc or in a video file. On videodisc, frames are numbered sequentially from 1 to 54,000 on each side and can be accessed individually; on videotape, the numbers are assigned by way of the SMPTE time code.

frame rate. The speed at which the frames are scanned. For a videodisc player, the speed at which frames are scanned is 30 frames per second for NTSC video. For most videotape devices, the speed is 24 frames per second.

friend class. A class in which all the member functions are granted access to the private and protected members of another class. It is named in the declaration of the other class with the prefix friend.

friend function. A function that is granted access to the private and protected parts of a class. It is named in the declaration of the class with the prefix friend.

full-motion video. (1) Video playback at 30 frames per second on NTSC signals. (2) A digital video compression technique that operates in real time.

G

gain. The ability to change the audibility of the sound, such as during a fade in or fade out of music.

GDI. Graphics device interface

graphic attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string.

graphical user interface (GUI). Type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop.

graphics. A picture defined in terms of graphic primitives and graphic attributes.

Graphics device interface object. An object such as a brush, pen, or bitmap. All graphics device interface (GDI) objects are owned by the process that also owns the thread. Compare to user object and kernel object.

GUI. Graphical user interface.

GUID. The globally unique identifier for an object. The system Registry uses GUIDs to distinguish all OLE objects available on a system. A CLSID is a type of GUID.

H

halftone. The reproduction of continuous-tone artwork, such as a photograph, by converting the image into dots of various sizes.

hash function. A function that determines which category, or bucket, to put an element in. A hash function is needed when implementing a hash table.

hash table. A data structure that divides all elements into (preferably) equal-sized categories, or buckets, to allow quick access to the elements. The hash function determines which bucket an element belongs in.

header file. A file that can contain system-defined control information or user data and generally consists of declarations.

heap. An unordered flat collection that allows duplicate elements.

height of a tree. The length of the longest path from the root to a leaf.

hit testing. The means of identifying which graphic object the mouse is pointing to.

I

implementation class. A class that implements a concrete class. Implementation classes are never used directly.

incomplete class declaration. A class declaration that does not define any members of a class. Typically, you use an incomplete class declaration as a forward declaration.

indirection. A mechanism for connecting objects by storing, in one object, a reference to another object.

inheritance. (1) A mechanism by which a derived class can use the attributes, relationships, and member functions defined in more abstract classes related to it (its base classes). See also multiple inheritance. (2) An object-oriented programming technique that allows you to use existing classes as bases for creating other classes.

initializer. An expression used to initialize objects.

inlined function •keyword

inlined function. A function call that the compiler replaces with the actual code for the function. You can direct the compiler to inline a function with the inline keyword.

in-place activation. The merging of the elements of user interfaces for a container and a server, such as small child windows, tool bars and menus, into the container's window space. The merger allows the user access to the server's controls from within the container, thereby providing a more document-centric approach to working. By contrast, simple activation of a server is a more application-centric approach to application interaction. Synonym for in situ editing.

input stream. A stream used to read input.

instance number. A number that the operating system uses to keep track of all of the instances of the same type of device. For example, the amplifier-mixer device name is AMPMIX plus a 2-digit instance number. If a program creates two amplifier-mixer objects, the device names could be AMPMIX01 and AMPMIX02.

integral object. A character object, an object having an enumeration type, an object having variations of the type int, or an object that is a bit field.

interactive graphics. Graphics that a user at a terminal can move or manipulate.

interactive video. The process of combining video and computer technology so that the user's actions, choices, and decisions affect the way in which the program unfolds.

interrupt. A temporary suspension of a process caused by an external event, performed in such a way that the process can be resumed.

intersection. Given collections A and B, the set of elements that is contained in both A and B.

intrinsic function. A function supplied by a program as opposed to a function supplied by the compiler.

inverted colors. Opposite colors in the light spectrum.

iteration. The process of repeatedly applying a function to a series of elements in a collection until some condition is satisfied.

iteration order. The order in which elements are accessed when iterating over a collection. In ordered collections, the element at position 1 will be accessed first, then the element at position 2, and so on. In sorted collections, the elements are accessed according to the ordering relation provided for

the element type. In collections that are not ordered the elements are accessed in an arbitrary order. Each element is accessed exactly once.

iterator class. A class that provides iteration functions.

I/O Stream Library. A class library that provides the facilities to deal with many varieties of input and output.

K

kernel. The core of an operating system, usually responsible for basic I/O and process execution.

kernel object. In Windows, an object used by the system and your applications to manage numerous resources, such as processes, threads, and files.

Compare to graphics device interface object and user object.

key access. A property that allows elements to be accessed by matching keys.

key bag. An unordered flat collection that uses keys and can contain duplicate elements.

key collection. (1) An abstract class that has the property of key access. (2) In general, any collection that uses keys.

key equality. A relation that determines whether two keys are equal.

key() function. When used on a flat collection, a function that returns a reference to the key of an element.

key-type function. Any of several functions of an element type, that are used by the Collection Class Library member functions to manipulate the keys of a class.

key set. An unordered flat collection that uses keys and does not allow duplicate elements.

key sorted bag. A sorted flat collection that uses keys and allows duplicate elements.

key sorted collection. An abstract class with the properties of key equality and sorted elements.

key sorted set. A sorted flat collection that uses keys and does not allow duplicate elements.

keyword. (1) A predefined word reserved for the C or C++ language that you cannot use as an identifier. (2) A symbol that identifies a parameter.

last element • mount

L

last element. The element accessed last in an iteration over a collection. Each collection has its own definition for last element. For example, the last element of a sorted set is the element with the largest value.

latched. The state of a button. A button in its latched state is held in its pressed position until the user clicks on it to release (unlatch) it.

leaves. In a tree, nodes without children. Synonymous with terminals.

library. (1) A collection of functions, function calls, subroutines, or other data. (2) A set of object modules that can be specified in a link command.

linkage editor. Synonym for linker.

linked component. A component whose data is not stored directly in the compound document itself. Instead, data is stored elsewhere, and the compound document includes a name that names the other location.

linked implementation. An implementation in which each element contains a reference to the next element in the collection. Pointer chains are used to access elements in linked implementations. Linked implementations are also called linked list implementations.

linked sequence. A sequence that uses a linked implementation.

linker. A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single executable program.

locale. The definition of the subset of a user's environment that depends on language and cultural conventions.

lvalue. An expression that represents an object that can be both examined and altered.

M

manipulator. A value that can be inserted into streams or extracted from streams to affect or query the behavior of the stream.

manual-reset event. In Windows, an event used to signal several threads simultaneously that an operation has completed.

See also event.

mask. A pattern of bits or characters that controls the keeping, deleting, or testing of portions of another pattern of bits or characters.

MBCS (multibyte character set). See multibyte character set

MDI. See multiple document interface.

member. Data, functions, or types contained in classes, structures, or unions.

member function. An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and member functions of objects of its class.

message. A request from one object that the receiving object implement a method. Because data is encapsulated and not directly accessible, a message is the only way to send data from one object to another. Each message specifies the name of the receiving object, the method to be implemented, and any parameters the method needs for implementation.

method. Synonym for member function.

MIDI. Musical Instrument Digital Interface. A standard used in the music industry for interfacing digital musical instruments.

mix. (1) An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as foreground mix. Contrast with background mix. (2) The combination of audio or video sources during postproduction.

mixer. A device used to simultaneously combine and blend several inputs into one or two outputs.

mode. A collection of attributes that specifies a file's type and its access permissions.

model. The data portion of a document component. The model and the view comprise the two pieces of a document component. The Compound Document Framework provides an IModel base class from which you can derive your own model classes.

motion video. Video that displays real motion.

mount. (1) To place a data medium in a position to operate. (2) To make recording media accessible.

Moving Pictures Experts Group (MPEG) •nonreentrant

Moving Pictures Experts Group (MPEG). (1) A group that is working to establish a standard for compressing and storing motion video and animation in digital form. (2) The compression standard of video and audio data that is stored on mass media.

MPEG. Moving Pictures Experts Group.

multibyte character set (MBCS). A character set whose characters consist of more than 1 byte. Used in languages such as Japanese, Chinese, and Korean, where the 256 possible values of a single-byte character set are not sufficient to represent all possible characters.

multimedia. Computer-controlled presentations combining any of the following: text, graphics, animation, full-motion images, still video images, and sound.

multiple document interface (MDI). An interface that uses a primary window to contain related document windows. The parent window's title bar is displayed along with the child window's title bar. If the child window displays a document window, an icon that indicates the application data's file type appears in the child window's title bar.

multiple inheritance. (1) An object-oriented programming technique implemented in C++ through derivation, in which the derived class inherits members from more than one base class. (2) The structuring of inheritance relationships among classes so a derived class can use the attributes, relationships, and functions used by more than one base class.

See also inheritance and class lattice.

multitasking. (1) A mode of operation that allows concurrent performance or interleaved execution of more than one task or program. (2) A process that allows a computer or operating system to run multiple applications or tasks concurrently by dividing the processor's time between them rapidly.

See also preemptive multitasking. Contrast with nonpreemptive multitasking.

multithread. Pertaining to concurrent operation of more than one path of execution within a computer.

multithreading. A process that allows a multitasking operating system to multitask subportions (threads) of an application smoothly.

mutex. (1) In Windows, a flag that prevents threads from interacting with the 16-bit kernel when another thread is executing code there. See also nonreentrant. (2) A synchronization kernel object that synchronizes data access across multiple processes. A mutex object is either signaled

or nonsignaled and is owned by a thread. See also critical section, semaphore, event, signaled, and nonsignaled.

N

n-ary tree. A tree that has an upper limit, n , imposed on the number of children allowed for a node.

National Television Standard Committee (NTSC). (1) A committee that sets the standard for color television broadcasting and video in the United States (currently in use also in Japan). (2) The standard set by the NTSC committee (the NTSC standard).

native. The rendering mechanism and format (RMF) that best represents the object and is the best one for rendering.

For example, a native of Cincinnati understands the streets in the area better than someone who has just moved there. Therefore, a Cincinnati native can get from point A to point B quicker than a newcomer. Likewise, a native RMF can get the data transferred from point A to point B more efficiently than the additional RMFs. We can use additional RMFs when we cannot use the native, or optimal, approach.

nested class. A class defined within the scope of another class.

new. (1) A C++ keyword identifying a free storage allocation operator. (2) A C++ operator used to create class objects.

new-line character. A control character that causes the print or display position to move to the first position on the next line. This control character is represented by `\n` in the C language.

node. In a tree structure, a point at which subordinate items of data originate.

nonpreemptive multitasking. A type of multitasking on 16-bit Windows where one application must notify the operating system that it is finished processing before the scheduler can assign another application execution time. Also called cooperative multitasking.

Contrast with preemptive multitasking.

nonreentrant. The state of 16-bit code in the kernel where two threads cannot access it at the same time without risking a system crash.

In Windows 95, processes are preempted and any thread is likely to be interrupted at any point in its execution.

See also mutex.

nonsignaled •owner window

nonsignaled. In Windows, the state that an object is in when it is suspended, or asleep. For example, when a thread is created and running, its associated thread kernel object is nonsignaled. As soon as the thread terminates, its thread kernel object is signaled.

notification area. (1) In Visual Builder, the information or status area at the bottom of the window. (2) In Windows 95, an area to the extreme right of the taskbar. By default (on machines with a sound driver), the notification area has two objects: a loudspeaker icon and text that shows the current time.

NTFS. See NT file system.

NT file system (NTFS). A Windows NT disk drive file system that restores disk-based data after a system failure. NTFS can manipulate extremely large storage media and has file names up to 255 characters in length.

NTSC. National Television Standard Committee.

NTSC format. The specifications for color television as defined by the NTSC, which include: (a) 525 scan lines, (b) broadcast bandwidth of 4 megaHertz, (c) line frequency of 15.75 kiloHertz, (d) frame frequency of 30 frames per second, and (e) color subcarrier frequency of 3.58 megaHertz.

null character ( ). The ASCII or EBCDIC character with the hex value 00 (all bits turned off).

O

object. (1) (2) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (3) A collection of data and member functions that operate on that data, which together represent a logical entity in the system. In object-oriented programming, objects are grouped into classes that share common data definitions and member functions. Each object in the class is said to be an instance of the class. (4) In Visual Builder, an instance of an object class consisting of attributes, a data structure, and operational member functions. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and member functions as other instances of the object class, though it has unique values assigned to its attributes. (5) In Windows, any item that is or can be linked into another Windows application, such as a sound, graphic, piece of text, or portion of a spreadsheet. An object must be from an application that supports OLE. See object linking and embedding (OLE).

object-oriented programming. A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on what data objects comprise the problem and how they are manipulated, not on how something is accomplished.

object linking and embedding (OLE). (1) An API that supports compound documents, cross-application macro control, and common object registration. OLE defines protocols for visual editing, drag-and-drop data transfers, structured storage, custom controls, and more. (2) A data sharing scheme that allows dissimilar applications to create single complex documents through a cooperative scheme. The documents can consist of material that a single application could not have created on its own.

OLE. See object linking and embedding (OLE).

operation class. A class that defines all required element and key operations required by a specific collection implementation.

operator function. An overloaded operator that is either a member of a class or that takes at least one argument that is a class type or a reference to a class type. See overloading.

optical reflective disc. An optical videodisc that is read by means of the reflection of a laser beam from the shiny surface on the disc.

ordered collection. (1) An abstract class that has the property of ordered elements. (2) In general, any collection that has its elements arranged so that there is always a first element, last element, next element, and previous element.

ordering relation. A property that determines how the elements are sorted. Ascending order is an example of an ordering relation.

overflow. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overloading. An object-oriented programming technique where one or more function declarations are specified for a single name in the same scope.

owner window. A window similar to a parent window, but it does not affect the behavior or appearance of the window. The owner coordinates the activity of a window.

P

pad. To fill unused positions in a field with data, usually 0's, 1's, or blanks.

parameter declaration. A description of a value that a function receives. A parameter declaration determines the storage class and the data type of the value.

parent node. A node to which one or more other nodes are subordinate.

parent window. A window that provides the child window information on how and where to draw it. The parent window also defines the relationship that the child window has with other windows in the system.

pause. To temporarily halt the medium. The halted visual should remain displayed but no audio should be played.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for pixel and picture element.

picture element. Synonym for pel.

pitch. The ability to change the key or keynote of the sound. For example, in music, the different pitches of people's voices are soprano, alto, tenor, baritone, and bass, arranged from the highest to lowest pitch.

pixel. Picture element. Synonym for pel.

pointer. A variable that holds the address of a data object or function.

pointer class. A class that implements pointers.

pointer to member. An operator used to access the address of nonstatic members of a class.

polymorphic function. A function that can be applied to objects of more than one data type. C++ implements polymorphic functions in two ways:

1. Overloaded functions (calls are resolved at compile time)
2. Virtual functions (calls are resolved at run time)

polymorphism. The technique of taking an abstract view of an object or function and using any concrete objects or arguments that are derived from this abstract view.

positioning property. The property of an element that is used to position the element in a collection. For example, the value of the key may be used as the positioning property.

precondition. A condition that a function requires to be true when it is called.

predicate function. A function that returns an IBoolean value of *true* or *false*. (IBoolean is an integer-represented Boolean type.)

preemptive multitasking. The operating system's ability to interrupt a thread at (almost) any time and assign the processor to a waiting thread. Multiple applications can thus run simultaneously, and a single application cannot control all of the system resources.

Contrast with nonpreemptive multitasking.

preparation. Any activity that the source performs before rendering the data. For example, the drag item may require that the source create a secondary thread for the source rendering to take place in. The system remains responsive to users so that they can do other tasks.

preprocessor. A phase of the compiler that examines the source program for preprocessor statements, which are then executed, resulting in the alteration of the source program.

preroll. To prepare a device to begin a playback or recording function with minimal delay.

primary thread. The first thread created when a process is initialized.

See also thread.

primitive. See graphic primitive.

primitive attribute. A specifiable characteristic of a graphic primitive. See graphic attributes.

priority queue. A queue that has a priority assigned to its elements. When accessing elements, the element with the highest priority is removed first. A priority queue has a largest-in, first-out behavior.

private. Pertaining to a class member that is accessible only to member functions and friends of that class.

process. (1) A collection of code, data, and other system resources, including at least one thread of execution, that performs a data processing task. (2) A running application, its address space, and its resources. (3) An instance of a running program. A Win32 process owns a 4-GB address space containing the code and data for an application's .exe file; it does not execute anything. It also owns certain resources, such as files, dynamic memory allocations, and threads. (4) A program running under OS/2, along with the resources associated with it (memory, threads, file system resources, and so on).

profiling •scope

profiling. The process of generating a statistical analysis of a program that shows processor time and the percentage of program execution time used by each procedure in the program.

program. (1) One or more files containing a set of instructions conforming to a particular programming language syntax. (2) A self-contained, executable module. Multiple copies of the same program can be run in different processes.

Program Files. A folder, which Windows 95 Setup places off the root of the drive on which Windows 95 is installed. It makes a subfolder called Accessories and places files such as WordPad and Paint there. This is the recommended default location for application programs.

project. A container that groups related objects (tasks) into a primary window. When the user opens the object, the object has its own primary window.

property function. A function that is used to determine whether the element it is applied to has a given property or characteristic. A property function can be used, for example, to remove all elements with a given property.

protected. Pertaining to a class member that is only accessible to member functions and friends of that class, or to member functions and friends of classes derived from that class.

prototype. A function declaration or definition that includes both the return type of the function and the types of its arguments.

public. Pertaining to a class member that is accessible to all functions.

pure virtual function. A virtual function that has a function initializer of the form `= 0;`.

Q

queue. A sequence with restricted access in which elements can only be added at the back end (or bottom) and removed from the front end (or top). A queue is characterized by first-in, first-out behavior and chronological order.

R

reference class. A class that links a concrete class to an abstract class. Reference classes make polymorphism possible with the Collection Classes.

relation. An unordered flat collection class that uses keys, allows for duplicate elements, and has element equality.

renderer. An object that renders data using a particular mechanism, such as using files or shared memory. It contains definitions of supported rendering mechanisms and formats and types. Renderers are maintained positionally (1-based).

rendering. The transfer or re-creation of the dragged object from the source window to the target window.

rendering format. Identifies the actual format of the data being rendered in a direct manipulation operation.

rendering mechanism. Identifies the actual format of the data being rendered in a direct manipulation operation.

resource file. A file that contains data used by an application, such as text strings and icons.

returned element. An element returned by a function as the return value.

RGB. Red, green, blue. A method of processing color images according to their red, green, and blue color content.

RMFs. Rendering mechanisms and formats.

root. A node that has no parent. All other nodes of a tree are descendants of the root.

S

samples-per-second. The number of times per second that the audio card records data from the audio input. For example, 44 kiloHertz is CD quality; 22 kiloHertz is FM music quality; and 11 kiloHertz is voice quality.

SBCS (Single-Byte Character Set). See single-byte character set

scan. To search backward and forward at high speed on a CD audio device. Scanning is analogous to fast forwarding.

scope. That part of a source program in which an object is defined and recognized.

scope operator (::) •stream

scope operator (::). An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.

scroll increment. The number by which the current value of the circular slider is incremented or decremented when a user presses one of the circular slider control buttons.

semaphore. A synchronization kernel object used for resource-counting. A semaphore offers a thread the ability to query the number of resources available. If one or more resources are available, the count of available resources is decremented.

See also critical section, mutex, and event.

sequence. A sequentially ordered flat collection.

sequential collection. An abstract class with the property of sequentially ordered elements.

server. An application, or document component, that supplies an object, as opposed to a container, which contains objects. For example, a drawing program that provides a picture that can be placed inside a word-processing document is referred to as a server.

siblings. All the children of a node are said to be siblings of one another.

signaled. A state in which an object has been reactivated after the threads have been put to sleep. For example, if a thread in a parent process needs to wait for the child process to terminate, the parent's thread puts itself to sleep until the kernel object identifying the child process becomes signaled.

single-byte character set (SBCS). A set of characters in which each character is represented by a 1-byte code.

SMPTE time code. A frame-numbering system developed by SMPTE that assigns a number to each frame of video. The 8-digit code is in the form HH:MM:SS:FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point.

sorted bag. A sorted flat collection that allows duplicate elements.

sorted collection. (1) An abstract class with the property of sorted elements. (2) In general, any collection with sorted elements.

sorted map. A sorted flat collection with key and element equality.

sorted relation. A sorted flat collection that uses keys, has element equality, and allows duplicate elements.

sorted set. A sorted flat collection with element equality.

sound track. Synonymous with audio track.

sprite. A small graphic that can be moved independently around the screen, producing animated effects.

stack. A data structure in which new elements are added to and removed from the top of the structure. A stack is characterized by Last-In-First-Out (LIFO) behavior.

standard error. An output stream usually intended to be used for diagnostic messages.

standard input. An input stream usually intended to be used for primary data input. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. An output stream usually intended to be used for primary data output. When programs are run interactively, standard output usually goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

step backward. In multimedia applications, to move the medium backward one frame or segment at a time.

step forward. In multimedia applications, to move the medium forward one frame or segment at a time.

step frame. A function of devices such as digital video and videodisc players that enables a user to move frame-by-frame in either direction.

storage. Like a directory in a conventional file system, storage manages other storages and streams but holds no data itself. Storage objects work with stream objects to provide persistent storage. A stream acts like a file in that it can hold information but not other storage elements.

stream. (1) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. (2) A file access object that allows access to an ordered sequence of characters, as described by the ISO C standard. A stream provides the additional services of user-selectable buffering and formatted input and output. (3) Like a single disk file in a conventional file system, a stream is kept in a storage object, which is like a directory in a conventional file system.

stream buffer •tree

A stream is named using a text string, and it can have any internal structure that you define.

stream buffer. A stream buffer is a buffer between the ultimate consumer, ultimate producer, and the I/O Stream Library functions that format data. It is implemented in the I/O Stream Library by the `streambuf` class and the classes derived from `streambuf`.

string. A contiguous sequence of characters.

structure. A construct that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

structured storage. A standardized means of accessing units of information using storages and streams. Conceptually, structured storage is like a file system within a file that allows diverse collections of data to be stored in a single file.

subclass. See derived class.

subscript. One or more expressions, each enclosed in brackets, that follow an array name. A subscript refers to an element in an array.

subtree. A tree structure created by arbitrarily denoting a node to be the root node in a tree. A subtree is always part of a whole tree.

superclass. See base class and abstract class.

superset. Given two sets A and B, A is a superset of B if and only if all elements of B are also elements of A. That is, A is a superset of B if B is a subset of A.

T

tabular implementation. An implementation that stores the location of elements in tables. Elements in a tabular implementation are accessed by using indices to arrays.

tabular sequence. A sequence that uses a tabular implementation.

taskbar. In Windows 95, a bar at the bottom (default position) of the desktop that shows all open applications and active windows. Clicking on any task button brings the corresponding session to the foreground.

template. A family of classes or functions where the code remains invariant but operates with variable types.

terminals. Synonym for *leaves*.

this. A C++ keyword that identifies a special type of pointer in a member function, one that references the class object with which the member function was invoked.

this collection. The collection to which a function is applied.

thread. (1) The atomic unit or path of execution within a process, a piece of executing code. (2) In Windows, each thread is located in its own stack from the owning process' 4-GB address space, and each one has its own set of processor registers, called the thread's context. See also primary thread and zero page thread.

thread synchronization. The ability to synchronize the activities of various threads. A thread synchronizes itself with another thread by putting itself to sleep. Before doing so, the thread notifies the operating system as to what event has to occur in order for the thread to resume execution.

throw expression. An argument to the C++ exception being thrown.

time code. See SMPTE time code.

tool bar. The area under the title bar that displays the tools available.

transparency. Refers to when a selected color on a graphics screen is made transparent to allow the video behind it to become visible.

transparent color. (1) A clear color used to indicate the part of the bitmap that is not drawn for the bitmap. The area under the bitmap is not overpainted for areas of the bitmap that are set to the transparent color. (2) Video information is considered as being present on the video plane that is maintained behind the graphics plane. When an area on the graphics plane is painted with a transparent color, the video information in the video plane is made visible.

trap. An unprogrammed conditional jump to a specified address that is automatically activated by hardware. A recording is made of the location from which the jump occurred.

treble. (1) The upper half of the whole vocal or instrumental tonal range. (2) The higher portion of the audio frequency range in sound recording.

tree. A hierarchical collection of nodes that can have an arbitrary number of references to other nodes. A unique path connects every two nodes.

tree control • virtual function

tree control. A type of control that shows the hierarchical relationships among a set of objects by indenting them as in an outline. The user can expand or collapse the various branches (levels) of the tree (outline).

true and additional. The most accurate or most descriptive (primary) type of an object (true) and the other or secondary types (additional). For example, if the object is a text file, its true type is text; if the file was a C source code file, its true type is C code.

try block. A block in which a known C++ exception is passed to a handler.

typed implementation class. A class that implements a concrete class and provides an interface that is specific to a given element type. This interface allows the compiler to verify that, for example, integers cannot be added to a set of strings.

typeless implementation class. A class that implements a concrete class and provides an interface that is not specific to a given element type.

U

ultimate consumer. The target of data in an I/O operation. An ultimate consumer can be a file, a device, or an array of bytes in memory.

ultimate producer. The source of data in an I/O operation. An ultimate producer can be a file, a device, or an array of bytes in memory.

unbounded collection. A collection that has no upper limit on the number of elements it can contain.

undefined cursor. A cursor that may or may not be valid, and that may or may not refer to a different element of the collection from the element it referred to before the function call that resulted in its becoming undefined. An undefined cursor may refer to no element of the collection, and still be a valid cursor.

underflow. (1) A condition that occurs when the result of an operation is less than the smallest possible nonzero number. (2) Synonym for arithmetic underflow, monadic operation.

union. (1) Structures that can contain different types of objects at different times. Only one of the member objects can be stored in a union at any time. (2) Given the sets A and B, all elements of A, B, or both A and B.

unique collection. A collection in which the value of an element only occurs once; that is, there are no duplicate elements.

unload. To eject the medium from the device.

unordered collection. A collection that has no order to its elements.

user object. An object, such as an icon, a window, a menu, or an accelerator table. In Windows, most user objects are owned by the thread that created them (icons, cursors, and windows classes are owned by a process).

V

VCR. Videocassette recorder.

VGA. Video graphics adapter.

view. The user interface controls and associated handlers for a document component. The view and the model comprise the two pieces of a document component. The Compound Document Framework provides an IView base class from which you can derive your own view classes.

video. Pertaining to the portion of recorded information that can be seen.

video attributes. The standard video attributes are: brightness, contrast, freeze, hue, saturation, and sharpness.

video graphics adapter (VGA). A graphics controller for color displays. The pixel resolution of the video graphics adapter is 4:4.

videocassette recorder (VCR). A device for recording or playing back videocassettes.

videodisc. A disc on which programs have been recorded for playback on a computer or a television set; a recording on a videodisc. The most common format in the United States and Japan is an NTSC signal recorded on the optical reflective format.

videodisc player. A device that provides video playback for prerecorded videodiscs.

virtual function. A function of a class that is declared with the keyword virtual. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called. This is determined at run time.

volatile •(::) (double colon)

volatile. An attribute of a data object that indicates the object is changeable beyond the control or detection of the compiler. Any expression referring to a volatile object is evaluated immediately, for example, assignments.

volume. The intensity of sound. A volume of 0 is minimum volume. A volume of 100 is maximum volume.

W

white space. Space characters, tab characters, form feed characters, and new-line characters.

wide character. A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales.

Win32. The name of an application programming interface (API).

See also Win32 API.

Win32 API. (1) A set of Win32 functions that can be called from source code. (2) A 32-bit version of the 16-bit Windows 3.1 API (native to Windows NT).

See also Win32.

Win32s. A platform that the Win32 API is implemented on. (The s stands for subset.) It consists of a virtual-device driver and dynamic link libraries (DLLs) that add the Win32 API to the 16-bit Windows 3.n system. It includes structured exception handling and limited implementations of memory-mapped files.

See also Win32 API and Windows 95.

Windows NT. A platform that the Win32 API is implemented on. It is a portable, high-end operating system, which can run several different types of applications simultaneously. It is the only Win32 platform for machine architectures based on processors other than the x86, and it supports multiple processors.

See also Win32 API.

Windows 95. (1) A 32-bit operating system that allows you to run 32-bit application. Windows 95 is a multitasking, multithreaded operating system that can control multiple programs at once. Each program can have multiple concurrent threads or independently executing subcomponents. (2) A platform that the Win32 API is implemented on. It supports image color matching, modems, and other services. It partially supports asynchronous file I/O, debugging, registry, security, and event-logging functions.

workspace. A container object that provides for the association and management of task-related windows within a parent window.

Z

zero page thread. A thread with a priority level of 0 that zeros out any free pages in the system when there are no other threads that need to perform work in the system. No other threads can have a priority level of 0.

See also thread and primary thread.

Numerics

24-bit color. A digital standard that uses 24 bits of information to describe each color pixel, providing up to 16.7 million colors in one image (the highest digital standard currently available).

8-bit color. A digital standard that uses 8 bits of information to describe each color pixel, providing up to 256 colors in one image (the standard for VGA displays).

Special Characters

(::) (double colon). Scope operator. An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.



Bibliography

This bibliography lists the publications that make up the IBM VisualAge for C++ library and related publications. The list of related publications is not exhaustive but should be adequate for most VisualAge for C++ users.

The IBM VisualAge for C++ Library

The following books are part of the IBM VisualAge for C++ library.

- *Installation Guide & Product Overview*, S33H-5030
- *User's Guide*, S33H-5031
- *Programming Guide*, S33H-5032
- *Visual Builder User's Guide*, S33H-5034
- *Visual Builder Parts Reference*, S33H-5035
- *Building VisualAge for C++ Parts for Fun and Profit*, S33H-5036
- *Open Class Library User's Guide*, S33H-5033
- *Open Class Library Reference*, S33H-5039
- *Language Reference*, S33H-5037-00
- *C Library Reference*, S33H-5038
- *SOM Programming Guide*, S33H-5044
- *SOM Programming Reference*, SOM Programming Reference

C and C++ Related Publications

- *Portability Guide for IBM C*, SC09-1405
- *American National Standard for Information Systems / International Standards Organization — Programming Language C (ANSI/ISO 9899-1990[1992])*

Non-IBM Publications

Many books have been written about the C++ language and related programming topics. The authors use varying approaches and emphasis. The following is a sample of some

non-IBM C++ publications that are generally available. This sample is not an exhaustive list. IBM does not specifically recommend any of these books, and other C++ books may be available in your locality.

- *The Annotated C++ Reference Manual* by Margaret A. Ellis and Bjarne Stroustrup, Addison-Wesley Publishing Company.
- *C++ Primer* by Stanley B. Lippman, Addison-Wesley Publishing Company.
- *Object-Oriented Design with Applications* by Grady Booch, Benjamin/Cummings.
- *Object-Oriented Programming Using SOM and DSOM* by Christina Lau, Van Nostrand Reinhold..
- *OS/2 C++ Class Library: Power GUI Programming with C Set ++* by Kevin Leong, William Law, Robert Love, Hiroshi Tsuji, and Bruce Olson, John Wiley and Sons Publishing Company.

Suggested Reading for Collection

Classes: These books contain explanations of data structures that may help you understand the data structures in the Collection Classes:

- *Data Structures and Algorithms* by Aho, Hopcroft, and Ullman, Addison-Wesley Publishing Company.
- *The Art of Computer Programming, Vol. 3: Sorting and Searching*, D.E. Knuth, Addison-Wesley Publishing Company.
- *C++ Components and Algorithms* by Scott Robert Ladd, M&T Publishing Inc.
- *A Systematic Catalogue of Reusable Abstract Data Types* by Juergen Uhl and Hans Albrecht Schmit, Springer Verlag.



Index

Special Characters

- ~Cursor
 - IGList::Cursor 322
 - IMMAudioCDContents::Cursor 490
- ~FaceNameCursor
 - IFont::FaceNameCursor 270
- ~IComponent
 - IComponent 8
- ~IComponentStationery
 - IComponentStationery 19
- ~IComponentStationeryFor
 - IComponentStationeryFor 21
- ~IDMCnrItem
 - IDMCnrItem 36
- ~IDMEFIItem
 - IDMEFIItem 42
- ~IDMEvent
 - IDMEvent 47
- ~IDMHandler
 - IDMHandler 55
- ~IDMImage
 - IDMImage 64
- ~IDMItem
 - IDMItem 77
- ~IDMItemProvider
 - IDMItemProvider 103
- ~IDMItemProviderFor
 - IDMItemProviderFor 108
- ~IDMMenuItem
 - IDMMenuItem 111
- ~IDMMLEItem
 - IDMMLEItem 116
- ~IDMOperation
 - IDMOperation 128
- ~IDMRenderer
 - IDMRenderer 132
- ~IDMSourceBeginEvent
 - IDMSourceBeginEvent 137
- ~IDMSourceDiscardEvent
 - IDMSourceDiscardEvent 140
- ~IDMSourceEndEvent
 - IDMSourceEndEvent 142
- ~IDMSourceHandler
 - IDMSourceHandler 146
- ~IDMSourceOperation
 - IDMSourceOperation (continued)
 - IDMSourceOperation 154
- ~IDMSourcePrepareEvent
 - IDMSourcePrepareEvent 161
- ~IDMSourcePrintEvent
 - IDMSourcePrintEvent 165
- ~IDMSourceRenderer
 - IDMSourceRenderer 168
- ~IDMSourceRenderEvent
 - IDMSourceRenderEvent 175
- ~IDMTTargetDropEvent
 - IDMTTargetDropEvent 180
- ~IDMTTargetEndEvent
 - IDMTTargetEndEvent 184
- ~IDMTTargetEnterEvent
 - IDMTTargetEnterEvent 188
- ~IDMTTargetEvent
 - IDMTTargetEvent 192
- ~IDMTTargetHandler
 - IDMTTargetHandler 196
- ~IDMTTargetHelpEvent
 - IDMTTargetHelpEvent 203
- ~IDMTTargetLeaveEvent
 - IDMTTargetLeaveEvent 206
- ~IDMTTargetOperation
 - IDMTTargetOperation 208
- ~IDMTTargetRenderer
 - IDMTTargetRenderer 214
- ~IDMTBarButtonItem
 - IDMTBarButtonItem 221
- ~IDMTToolBarItem
 - IDMTToolBarItem 228
- ~IDocumentStorage
 - IDocumentStorage 233
- ~IEmbeddedComponent
 - IEmbeddedComponent 236
- ~IEmbedderModel
 - IEmbedderModel 244
- ~IFlatFileStorage
 - IFlatFileStorage 251
- ~IFont
 - IFont 257
- ~IG3PointArc
 - IG3PointArc 276
- ~IGArc
 - IGArc 282

- ~IGBitmap
 - IGBitmap 290
- ~IGChord
 - IGChord 303
- ~IGEllipse
 - IGEllipse 307
- ~IGLine
 - IGLine 310
- ~IGList
 - IGList 315
- ~IGPie
 - IGPie 327
- ~IGPolygon
 - IGPolygon 332
- ~IGPolyline
 - IGPolyline 335
- ~IGrabHandles
 - IGrabHandles 339
- ~IGraphic
 - IGraphic 347
- ~IGraphicBundle
 - IGraphicBundle 359
- ~IGraphicContext
 - IGraphicContext 382
- ~IGRectangle
 - IGRectangle 408
- ~IGRegion
 - IGRegion 412
- ~IGString
 - IGString 426
- ~IGUIBundle
 - IGUIBundle 434
- ~IMM24FramesPerSecondTime
 - IMM24FramesPerSecondTime 440
- ~IMM25FramesPerSecondTime
 - IMM25FramesPerSecondTime 443
- ~IMM30FramesPerSecondTime
 - IMM30FramesPerSecondTime 446
- ~IMMAmpMixer
 - IMMAmpMixer 449
- ~IMMAudioBuffer
 - IMMAudioBuffer 463
- ~IMMAudioCD
 - IMMAudioCD 472
- ~IMMAudioCDContents
 - IMMAudioCDContents 485
- ~IMMCDXA
 - IMMCDXA 494
- ~IMMConfigurableAudio
 - IMMConfigurableAudio 504
- ~IMMCuePointEvent
 - IMMCuePointEvent 509
- ~IMMDevice
 - IMMDevice 523
- ~IMMDeviceEvent
 - IMMDeviceEvent 547
- ~IMMDeviceHandler
 - IMMDeviceHandler 551
- ~IMMDeviceNotifyHandler
 - IMMDeviceNotifyHandler 555
- ~IMMDigitalVideo
 - IMMDigitalVideo 560
- ~IMMErrorInfo
 - IMMErrorInfo 581
- ~IMMFileMedia
 - IMMFileMedia 584
- ~IMMHourMinSecFrameTime
 - IMMHourMinSecFrameTime 594
- ~IMMHourMinSecTime
 - IMMHourMinSecTime 598
- ~IMMMasterAudio
 - IMMMasterAudio 602
- ~IMMMillisecondTime
 - IMMMillisecondTime 607
- ~IMMMinSecFrameTime
 - IMMMinSecFrameTime 610
- ~IMMNotifyEvent
 - IMMNotifyEvent 614
- ~IMMPassDeviceEvent
 - IMMPassDeviceEvent 617
- ~IMMPlayableDevice
 - IMMPlayableDevice 625
- ~IMMPlayerPanel
 - IMMPlayerPanel 634
- ~IMMPlayerPanelHandler
 - IMMPlayerPanelHandler 639
- ~IMMPositionChangeEvent
 - IMMPositionChangeEvent 643
- ~IMMRecordable
 - IMMRecordable 646
- ~IMMRemovableMedia
 - IMMRemovableMedia 658
- ~IMMRemovableMediaHandler
 - IMMRemovableMediaHandler 666
- ~IMMRemovableMediaNotifyHandler
 - IMMRemovableMediaNotifyHandler 670
- ~IMMSequencer
 - IMMSequencer 673
- ~IMMSpeed
 - IMMSpeed 678

- ~IMMTime
 - IMMTime 682
- ~IMMTrackMinSecFrameTime
 - IMMTrackMinSecFrameTime 691
- ~IMMWaveAudio
 - IMMWaveAudio 698
- ~IModel
 - IModel 706
- ~IStructuredStorage
 - IStructuredStorage 712
- ~ITransformMatrix
 - ITransformMatrix 717
- ~TView
 - IView 726
- ~PointSizeCursor
 - IFont::PointSizeCursor 273

A

- acquire
 - IMMDevice 528
- addAsFirst
 - IGList 314
- addAsLast
 - IGList 314
- addAsNext
 - IGList 314
- addAsPrevious
 - IGList 314
- addAtPosition
 - IGList 314
- addCuePoint
 - IMMPlayableDevice 620
- addEntryAsFirst
 - IMMAudioCDCContents 485
- addEntryAsNext
 - IMMAudioCDCContents 485
- addItem
 - IDMOperation 123
- addPoint
 - IGPolyline 336
- addRenderer
 - IDMHandler 57
 - IDMSourceHandler 147
 - IDMTargetHandler 197
- addRMF
 - IDMItem 90
- addToBoundingRect
 - IGraphicContext 380

- addType
 - IDMItem 88
- adopt
 - IEmbedderModel 244
- adoptComponent
 - IGUIBundle 429
- adoptFrameWindow
 - IGUIBundle 430
- adoptModel
 - IComponent 8
- adoptView
 - IGUIBundle 430
- Advanced Control, Dialog, and Handler Classes xvi
 - overview xvi
- aliasName
 - IMMDevice 523
- allDevices
 - IMMDevice 539
- allocateOperation
 - IDMSourceHandler 151
 - IDMTargetHandler 200
- allowInComingCalls
 - IComponentStationery 17
- alternateWindow
 - IDMSourcePrepareEvent 161
 - IDMSourceRenderEvent 175
 - IDMTargetEndEvent 183
- alternateWindowHandle
 - IDMSourcePrepareEvent 161
 - IDMSourceRenderEvent 175
 - IDMTargetEndEvent 184
- ampMixer
 - IMMDevice 540
- animation
 - IMMDevice 540
- any
 - IDM 24
- appendRMF
 - IDMItem 91
- appName
 - IComponentStationery 14
- area
 - IEmbeddedComponent 237
- areHeadphonesEnabled
 - IMMMasterAudio 602
- areSpeakersEnabled
 - IMMMasterAudio 603
- asIndex
 - IMMAudioCDCContents::Cursor 490

- asMATRIXLF
 - ITransformMatrix 717
- asMMTime
 - IMMTime 682
- aspect
 - IEmbeddedComponent 236
- asPointerHandle
 - IGBitmap 286
- asString
 - IMMHourMinSecFrameTime 594
 - IMMHourMinSecTime 599
 - IMMMinSecFrameTime 611
 - IMMTime 683
 - IMMTrackMinSecFrameTime 692
- asTagXFORM
 - ITransformMatrix 717
- attachTo
 - IEmbedderModel 247
 - IModel 706
- attributes
 - IDMItem 78
- audio
 - IMMAudioBuffer 464
- audioCD
 - IMMDevice 540
- audioTape
 - IMMDevice 540
- avgCharWidth
 - IFont 259
- avgLowercase
 - IFont 259
- avgUppercase
 - IFont 259

B

- backgroundColor
 - IGraphicBundle 356
 - IGraphicContext 392
- backgroundMixMode
 - IGraphicBundle 355
 - IGraphicContext 392
- backwardDiag1
 - IGraphicBundle 370
- backwardDiag2
 - IGraphicBundle 370
- balance
 - IMMAmpMixer 450
- bass
 - IMMAmpMixer 450

- begin
 - IDMSourceOperation 156
- beginUsingFont
 - IFont 257
- binary
 - IDM 24
- binaryData
 - IDM 24
- bitmap
 - IDM 24
 - IDMImage 64
- bitsPerSample
 - IMMAudioBuffer 460
 - IMMConfigurableAudio 499
- black
 - IGBitmap 296
- blockAlignment
 - IMMAudioBuffer 460
 - IMMConfigurableAudio 499
- bmp
 - IDMImage 68
- bottomGraphicUnderPoint
 - IGList 316
- boundingRect
 - IGraphic 344
- boundingRectAt
 - IGList 316
- bundle
 - IComponent 9
 - IComponentStationery 16
 - IEmbeddedComponent 238
 - IView 722
- bytesPerSecond
 - IMMAudioBuffer 460
 - IMMConfigurableAudio 499

C

- canBeCopied
 - IDMItem 84
- canBeLinked
 - IDMItem 84
- canBeMoved
 - IDMItem 84
- canRedo
 - IMMRecordable 647
- canRender
 - IDMSourceRenderer 169
 - IDMTargetRenderer 215

- canRetry
 - IDMSourceRenderEvent 176
- canUndo
 - IMMRecordable 647
- cdxa
 - IMMDevice 540
- centerPoint
 - IGLine 311
- changeTo
 - IGrabHandles 340
- channels
 - IMMAudioBuffer 460
 - IMMConfigurableAudio 500
- charWidth
 - IFont 259
- classDefaultStyle
 - IDMImage 68
- Classes
 - IComponent 7
 - IComponentStationery 13
 - IComponentStationeryFor 20
 - IDM 23
 - IDMCnrlItem 35
 - IDMEFlItem 41
 - IDMEvent 46
 - IDMHandler 48
 - IDMImage 60
 - IDMImage::Style 70
 - IDMItem 72
 - IDMItemProvider 102
 - IDMItemProviderFor 107
 - IDMMenuItem 110
 - IDMMLEItem 115
 - IDMOperation 120
 - IDMRenderer 131
 - IDMSourceBeginEvent 136
 - IDMSourceDiscardEvent 139
 - IDMSourceEndEvent 142
 - IDMSourceHandler 145
 - IDMSourceOperation 153
 - IDMSourcePrepareEvent 160
 - IDMSourcePrintEvent 164
 - IDMSourceRenderer 167
 - IDMSourceRenderEvent 174
 - IDMTargetDropEvent 179
 - IDMTargetEndEvent 183
 - IDMTargetEnterEvent 187
 - IDMTargetEvent 192
 - IDMTargetHandler 195
 - IDMTargetHelpEvent 202

Classes (*continued*)

- IDMTargetLeaveEvent 205
- IDMTargetOperation 207
- IDMTargetRenderer 213
- IDMTBarButtonItem 220
- IDMToolBarItem 227
- IDocumentStorage 231
- IEmbeddedComponent 235
- IEmbedderModel 243
- IFlatFileStorage 250
- IFont 253
- IFont::FaceNameCursor 269
- IFont::PointSizeCursor 272
- IG3PointArc 275
- IGArc 280
- IGBitmap 285
- IGChord 302
- IGEllipse 305
- IGLine 309
- IGList 313
- IGList::Cursor 321
- IGPie 325
- IGPolygon 330
- IGPolyline 334
- IGrabHandles 339
- IGraphic 343
- IGraphicBundle 353
- IGraphicContext 379
- IGRectangle 406
- IGRegion 410
- IGString 423
- IGUIBundle 429
- IMGrabbable 437
- IMM24FramesPerSecondTime 439
- IMM25FramesPerSecondTime 442
- IMM30FramesPerSecondTime 445
- IMMAmpMixer 448
- IMMAudioBuffer 459
- IMMAudioCD 467
- IMMAudioCDContents 484
- IMMAudioCDContents::Cursor 489
- IMMCDXA 493
- IMMConfigurableAudio 498
- IMMCuePointEvent 509
- IMMDevice 512
- IMMDeviceEvent 547
- IMMDeviceHandler 550
- IMMDeviceNotifyHandler 554
- IMMDigitalVideo 557
- IMMErrorInfo 579

Classes (*continued*)

- IMMFileMedia 584
- IMMHourMinSecFrameTime 592
- IMMHourMinSecTime 597
- IMMMasterAudio 601
- IMMMillisecondTime 606
- IMMMinSecFrameTime 609
- IMMNotifyEvent 613
- IMMPassDeviceEvent 617
- IMMPlayableDevice 620
- IMMPlayerPanel 631
- IMMPlayerPanelHandler 638
- IMMPositionChangeEvent 643
- IMMRecordable 646
- IMMRemovableMedia 657
- IMMRemovableMediaHandler 665
- IMMRemovableMediaNotifyHandler 669
- IMMSequencer 672
- IMMSpeed 677
- IMMTime 680
- IMMTrackMinSecFrameTime 689
- IMMWaveAudio 694
- IModel 703
- IRegionHandle 709
- IStructuredStorage 711
- ITransformMatrix 715
- IView 722
- clear
 - IGRegion 413
- clearClipRegion
 - IGraphicContext 403
- clippingRect
 - IGString 424
- clipRegion
 - IGraphicContext 403
- close
 - IMMAmpMixer 449
 - IMMDevice 526
- closed
 - IDMImage 68
- closeDoor
 - IMMRemovableMedia 658
- command
 - IMMNotifyEvent 613
 - IMMPlayerPanelHandler 640
- commandNotifyId
 - IMMDevice 542
- completion
 - IDMSourceRenderEvent 176
- IComponent 7

IComponent (*continued*)

- IDocumentStorage 232
- IGUIBundle 435
- IModel 705
- IView 723
- componentList
 - IEmbedderModel 245
- IComponentStationery 13
- IComponentStationeryFor 20
- Compound Document Framework Classes xvi
 - overview xvi
- compressedRMFs
 - IDMItem 91
- connectedDeviceId
 - IMMDevice 521
- container
 - IDM 24
 - IDMItem 98
 - IDMSourceBeginEvent 137
 - IDMTargetDropEvent 180
 - IDMTargetEnterEvent 188
- containerId
 - IDMCnrItem 37
 - IDMSourceBeginEvent 137
 - IDMTargetDropEvent 180
 - IDMTargetEnterEvent 188
- containerName
 - IDMItem 78
- containerObject
 - IDM 24
 - IDMOperation 121
- contains
 - IGraphic 346
- contents
 - IDMItem 83
 - IMMAudioCD 477
- contentsSize
 - IDMItem 83
- contentType
 - IMMAudioBuffer 463
- controlInformation
 - IMMAudioCDContents 486
- copy
 - IDMOperation 130
 - IGBitmap 286
 - IMMRecordable 647
- copyable
 - IDMItem 99
- copyFromBuffer
 - IMMWaveAudio 694

- copyToBuffer
 - IMMWaveAudio 695
- country
 - IMMAudioCDContents 486
- cpp files, description xvii
- CPP.NDX xviii
- CPP*.INF xviii
- CPPBRS.NDX xviii
- CPPWO*.DEF xviii
- CPPWO*.DLL xviii
- CPPWO*.LIB xviii
- CPPWO*.RSP xviii
- CPPWOC3.LIB xviii
- CPPWOC3U.MSG xviii
- createBundle
 - IComponentStationery 15
- createComponent
 - IComponentStationery 15
- createEmbeddedComponent
 - IEmbedderModel 245
- createFrameWindow
 - IComponentStationery 15
- createModel
 - IComponentStationery 15
 - IComponentStationeryFor 21
- createView
 - IComponentStationery 16
 - IComponentStationeryFor 21
- cueForPlayback
 - IMMPlayableDevice 624
- cueForRecording
 - IMMRecordable 648
- cuePoint
 - IMMDeviceHandler 551
- cuePointId
 - IMMDevice 542
- currentDrawingPosition
 - IGraphicContext 383
- currentFont
 - IGraphicContext 398
- Cursor
 - IGList::Cursor 321
 - IMMAudioCDContents::Cursor 489
- cut
 - IMMRecordable 648
- cutCopyBufferSize
 - IMMWaveAudio 695
- cutToBuffer
 - IMMWaveAudio 696

D

- dat
 - IMMDevice 540
- data
 - IMMAudioBuffer 461
- data member names, conventions xix
- DDE4C01E.MSG xviii
- debugSupport
 - IDMOperation 122
- defaultBackgroundColor
 - IGraphicContext 383
- defaultBackgroundMixMode
 - IGraphicContext 384
- defaultDragHandler
 - IDMHandler 49
- defaultDrawOperation
 - IGraphicContext 384
- defaultDropTargetHandler
 - IDMHandler 49
- defaultFillColor
 - IGraphicContext 384
- defaultFillPattern
 - IGraphicContext 384
- defaultMixMode
 - IGraphicContext 384
- defaultOperation
 - IDMTargetEnterEvent 189
- defaultPatternOrigin
 - IGraphicContext 385
- defaultPenColor
 - IGraphicContext 385
- defaultPenEndingStyle
 - IGraphicContext 385
- defaultPenJoiningStyle
 - IGraphicContext 385
- defaultPenPattern
 - IGraphicContext 385
- defaultPenType
 - IGraphicContext 385
- defaultPenWidth
 - IGraphicContext 386
- defaultSourceHandler
 - IDMHandler 49
- defaultSourceRenderer
 - IDMRenderer 133
- defaultStyle
 - IDMImage 67
- defaultTargetHandler
 - IDMHandler 49

- defaultTargetRenderer
 - IDMRenderer 133
- defaultTime
 - IMMTime 686
- deleteAll
 - IEmbedderModel 245
- deletePendingEvents
 - IMMDevice 525
- deleteRMF
 - IDMItem 91
- deleteSelection
 - IMMRecordable 649
- dense1
 - IGraphicBundle 371
- dense2
 - IGraphicBundle 371
- dense3
 - IGraphicBundle 371
- dense4
 - IGraphicBundle 371
- dense5
 - IGraphicBundle 372
- dense6
 - IGraphicBundle 372
- dense7
 - IGraphicBundle 372
- dense8
 - IGraphicBundle 372
- description
 - IMMDevice 524
- destinationRectangle
 - IMMDigitalVideo 568
- destInvert
 - IGBitmap 296
- device
 - IMMCuePointEvent 510
 - IMMDeviceEvent 548
 - IMMNotifyEvent 614
 - IMMPassDeviceEvent 618
 - IMMPositionChangeEvent 644
- deviceEvent
 - IMMDeviceHandler 551
- deviceEventId
 - IMMDevice 542
- deviceId
 - IMMDevice 523
- deviceName
 - IMMDevice 524
- deviceType
 - IMMDevice 524
- deviceType (*continued*)
 - IMMPlayerPanel 635
- deviceWindow
 - IMMDevice 535
- digitalVideo
 - IMMDevice 540
- Direct Manipulation Classes xvi
 - overview xvi
- direction
 - IGArc 282
- disableAudio
 - IMMDevice 513
- disableAutoPlay
 - IMMAudioCD 468
- disableConnector
 - IMMDevice 521
- disableContinuousPlay
 - IMMAudioCD 468
- disableHeadphones
 - IMMMasterAudio 602
- disableMonitoring
 - IMMAmpMixer 454
 - IMMDigitalVideo 566
- disableSpeakers
 - IMMMasterAudio 603
- discId
 - IMMAudioCD 468
 - IMMAudioCDContents 486
- discProgramKey
 - IMMAudioCD 482
- discTitle
 - IMMAudioCD 468
- discTitleKey
 - IMMAudioCD 482
- dispatchHandlerEvent
 - IDMSourceHandler 149
 - IDMTargetHandler 199
 - IMMDeviceHandler 552
 - IMMDeviceNotifyHandler 555
 - IMMRemovableMediaHandler 667
 - IMMRemovableMediaNotifyHandler 670
- IDM 23
- IDMCnrItem 35
- IDMEFIItem 41
- IDMEvent 46
- IDMHandler 48
- IDMImage 60
- IDMImage::Style 70
- IDMItem 72
- IDMItemProvider 102

- IDMItemProviderFor 107
- IDMMenuItem 110
- IDMMLItem 115
- IDMOperation 120
- IDMRenderer 131
- IDMSourceBeginEvent 136
- IDMSourceDiscardEvent 139
- IDMSourceEndEvent 142
- IDMSourceHandler 145
- IDMSourceOperation 153
- IDMSourcePrepareEvent 160
- IDMSourcePrintEvent 164
- IDMSourceRenderer 167
- IDMSourceRenderEvent 174
- IDMTargetDropEvent 179
- IDMTargetEndEvent 183
- IDMTargetEnterEvent 187
- IDMTargetEvent 192
- IDMTargetHandler 195
- IDMTargetHelpEvent 202
- IDMTargetLeaveEvent 205
- IDMTargetOperation 207
- IDMTargetRenderer 213
- IDMTBarItem 220
- IDMToolBarItem 227
- documentName
 - IComponent 9
- IDocumentStorage 231
- doDiscard
 - IDMSourceRenderer 171
- doDragDrop
 - IEmbeddedComponent 242
 - IMGrabbable 437
- doPrint
 - IDMSourceRenderer 171
- doRender
 - IDMSourceRenderer 172
- doRenderEnd
 - IDMSourceRenderer 172
- doRenderPrepare
 - IDMSourceRenderer 172
- drag
 - IDMOperation 130
- dragInfo
 - IDMOperation 129
- dragItem
 - IDMSourceEndEvent 143
 - IDMSourceRenderEvent 176
 - IDMTargetEndEvent 184
- dragWasInterrupted
 - IDMOperation 129
- draw
 - IEmbeddedComponent 239
 - IGrabHandles 341
 - IGraphicContext 389
 - IView 725
- drawContents
 - IView 725
- drawOn
 - IG3PointArc 278
 - IGArc 282
 - IGBitmap 291
 - IGChord 303
 - IGEllipse 307
 - IGLine 311
 - IGList 315
 - IGPie 327
 - IGPolygon 332
 - IGPolyline 337
 - IGraphic 344
 - IGRectangle 408
 - IGRegion 413
 - IGString 426
- drawOperation
 - IGraphicBundle 359
 - IGraphicContext 392
- dropIndicator
 - IDMTargetEnterEvent 189
- dropOffset
 - IDMTargetOperation 208
- dropPosition
 - IDMTargetDropEvent 180
 - IDMTargetOperation 208
- dropStatus
 - IDMItem 77
- Dynamic Data Exchange Classes xvi
 - overview xvi

E

- element11
 - ITransformMatrix 718
- element12
 - ITransformMatrix 718
- element21
 - ITransformMatrix 718
- element22
 - ITransformMatrix 718

- element31
 - ITransformMatrix 718
- element32
 - ITransformMatrix 718
- IEmbeddedComponent 235
- IEmbedderModel 243
- enableAudio
 - IMMDevice 514
- enableAutoPlay
 - IMMAudioCD 469
- enableBusyDialog
 - IComponentStationery 17
- enableConnector
 - IMMDevice 522
- enableContinuousPlay
 - IMMAudioCD 469
- enableCopy
 - IDMItem 84
- enableDataUpdate
 - IMMFileMedia 590
- enableDragDropFor
 - IDMHandler 50
- enableDragFrom
 - IDMHandler 52
- enableDropOn
 - IDMHandler 53
- enableHeadphones
 - IMMMasterAudio 602
- enableLink
 - IDMItem 84
- enableMonitoring
 - IMMAmpMixer 454
 - IMMDigitalVideo 566
- enableMove
 - IDMItem 85
- enableNotification
 - IMMDevice 525
 - IMMRemovableMedia 660
- enableNotRespondingDialog
 - IComponentStationery 17
- enableSpeakers
 - IMMMasterAudio 603
- enclosingRect
 - IGArc 282
 - IGEllipse 307
 - IGPie 327
 - IGRectangle 408
- endingPoint
 - IG3PointArc 277
 - IGLine 311

- endOfTrack
 - IMMAudioCDContents 486
- endUsingFont
 - IFont 257
- errorId
 - IMMErrorInfo 581
 - IMMNotifyEvent 614
- errorText
 - IMMNotifyEvent 614
- eventCode
 - IMMDeviceEvent 548
- eventData
 - IMMDeviceEvent 548
- externalLeading
 - IFont 259

F

- faceNameAt
 - IFont 257
- FaceNameCursor
 - IFont::FaceNameCursor 269
- fastForward
 - IMMPlayerPanelHandler 640
- fastForwardButton
 - IMMPlayerPanel 632
- fastSpeed
 - IMMDigitalVideo 563
- fattrs
 - IFont 254
- file
 - IDM 24
- fileExt
 - IComponentStationery 14
- fileName
 - IDocumentStorage 233
 - IMMFileMedia 585
- fileNormalSpeed
 - IMMDigitalVideo 563
- fillColor
 - IGraphicBundle 356
 - IGraphicContext 392
- filled
 - IGraphicBundle 373
- fillMode
 - IGPolygon 332
- fillPattern
 - IGraphicBundle 363
 - IGraphicContext 392

- finalize
 - IView 723
- findRendererFor
 - IDMSourceHandler 151
 - IDMTargetHandler 201
- findRenderersFor
 - IDMSourceHandler 151
 - IDMTargetHandler 201
- firstGraphic
 - IGList 317
- firstTimeEntered
 - IDMTargetOperation 210
- IFlatFileStorage 250
- IFont 253
 - IGString 426
- IFont::FaceNameCursor 269
- IFont::PointSizeCursor 272
- fontmetrics
 - IFont 255
- forwardDiag1
 - IGraphicBundle 373
- forwardDiag2
 - IGraphicBundle 373
- format
 - IMMAudioBuffer 463
 - IMMConfigurableAudio 500
 - IMMSpeed 678
- frames
 - IMMHourMinSecFrameTime 592
 - IMMMinSecFrameTime 609
 - IMMTrackMinSecFrameTime 689
- framesPerSecond
 - IMMHourMinSecFrameTime 592
- frameWindow
 - IGUIBundle 435
- function arguments, conventions xx
- function return types, conventions xix

G

- IG3PointArc 275
- gain
 - IMMAmpMixer 451
- GArc 280
- IGBitmap 285
- IGChord 302
- IGEllipse 305
- generateSourceItems
 - IDMCnrItem 37
 - IDMEFItem 43

- generateSourceItems (*continued*)
 - IDMItem 97
 - IDMMenuItem 112
 - IDMMLEItem 117
 - IDMTBarButtonItem 223
- generateSourceName
 - IDMItem 98
- getLocation
 - IGrabHandles 340
- getOutline
 - IGrabHandles 340
- getTableFromDisc
 - IMMAudioCD 481
- IGLine 309
- IGList 313
- IGList::Cursor 321
- global names, conventions xix
- goActive
 - IEmbeddedComponent 236
- goInactive
 - IEmbeddedComponent 237
- goToEntry
 - IMMAudioCD 472
- IGPie 325
- IGPolygon 330
- IGPolyline 334
- grabHandles
 - IEmbeddedComponent 238
- IGraphic 343
- graphicAt
 - IGList 317
- graphicAtPosition
 - IGList 317
- graphicBundle 353
 - IGraphic 344
 - IGraphicContext 392
- IGraphicContext 379
- IGRectangle 406
- IGRegion 410
- group
 - IDMItem 98
- IGString 423
- IGUIBundle 429

H

- h files, description xvii
- halftone
 - IGBitmap 297
 - IGraphicBundle 373

handle		hasBackgroundMixMode (<i>continued</i>)	
IGBitmap	286	IGraphicBundle	355
IGraphicContext	383	hasChanged	
IMMDigitalVideo	568	IEmbedderModel	247
handleConvert		IModel	704
IGUIBundle	430	hasDrawOperation	
handleCopy		IGraphicBundle	359
IGUIBundle	431	hasFillColor	
handleCut		IGraphicBundle	356
IGUIBundle	431	hasFillPattern	
handleDefaultMenuCommands		IGraphicBundle	363
IGUIBundle	431	hasFont	
handleDelete		IGraphicContext	398
IGUIBundle	431	IGString	426
handleDoVerb		hasGraphicBundle	
IGUIBundle	432	IGraphic	345
handleEditLinks		hasImage	
IGUIBundle	432	IDMItem	77
handleEventsFor		hasMixMode	
IMMDeviceHandler	551	IGraphicBundle	361
IMMRemovableMediaHandler	666	hasPatternOrigin	
handleExit		IGraphicBundle	361
IGUIBundle	432	hasPenColor	
handleFileNew		IGraphicBundle	356
IGUIBundle	432	hasPenEndingStyle	
handleFileOpen		IGraphicBundle	367
IGUIBundle	432	hasPenJoiningStyle	
handleFileSave		IGraphicBundle	369
IGUIBundle	433	hasPenPattern	
handleFileSaveAs		IGraphicBundle	363
IGUIBundle	433	hasPenType	
handleGetArea		IGraphicBundle	365
IComponent	11	hasPenWidth	
handleInsertObject		IGraphicBundle	366
IGUIBundle	433	hasTransformMatrix	
handleNotification		IGraphic	348
IView	726	hasTransparentColor	
handlePaste		IGBitmap	288
IGUIBundle	433	hasType	
handlePasteLink		IDMItem	88
IGUIBundle	433	hatched	
handlePasteSpecial		IGraphicBundle	374
IGUIBundle	434	hatchedDiag	
handleSelectAll		IGraphicBundle	374
IGUIBundle	434	headerData	
handleSetArea		IMMAudioBuffer	461
IComponent	11	headphone	
hasBackgroundColor		IMMDevice	540
IGraphicBundle	356	hitApertureSize	
hasBackgroundMixMode		IGraphicContext	399

- hitPoint
 - IGraphicContext 399
- hollow
 - IGraphicBundle 374
- horizontal
 - IGraphicBundle 374
- hours
 - IMMHourMinSecFrameTime 593
 - IMMHourMinSecTime 597
 - IMMTime 681
- hpp files, description xvii
- hundredths
 - IMMTime 681

I

- ibmcl.cat xix
- IComponent
 - IComponent 8
- IComponentStationery
 - IComponentStationery 19
- IComponentStationeryFor
 - IComponentStationeryFor 20
- icon
 - IDM 25
- id
 - IGraphic 347
- IDMCnrItem
 - IDMCnrItem 36
- IDMEFItem
 - IDMEFItem 42
- IDMEvent
 - IDMEvent 46
- IDMHandler
 - IDMHandler 55
- IDMImage
 - IDMImage 61
- IDMItem
 - IDMItem 74
- IDMItemProvider
 - IDMItemProvider 103
- IDMItemProviderFor
 - IDMItemProviderFor 108
- IDMMenuItem
 - IDMMenuItem 110
- IDMMLItem
 - IDMMLItem 115
- IDMOperation
 - IDMOperation 128
- IDMRenderer
 - IDMRenderer 132
- IDMSourceBeginEvent
 - IDMSourceBeginEvent 136
- IDMSourceDiscardEvent
 - IDMSourceDiscardEvent 139
- IDMSourceEndEvent
 - IDMSourceEndEvent 142
- IDMSourceHandler
 - IDMSourceHandler 145
- IDMSourceOperation
 - IDMSourceOperation 153
- IDMSourcePrepareEvent
 - IDMSourcePrepareEvent 161
- IDMSourcePrintEvent
 - IDMSourcePrintEvent 164
- IDMSourceRenderer
 - IDMSourceRenderer 167
- IDMSourceRenderEvent
 - IDMSourceRenderEvent 175
- IDMTargetDropEvent
 - IDMTargetDropEvent 179
- IDMTargetEndEvent
 - IDMTargetEndEvent 184
- IDMTargetEnterEvent
 - IDMTargetEnterEvent 188
- IDMTargetEvent
 - IDMTargetEvent 192
- IDMTargetHandler
 - IDMTargetHandler 195
- IDMTargetHelpEvent
 - IDMTargetHelpEvent 202
- IDMTargetLeaveEvent
 - IDMTargetLeaveEvent 205
- IDMTargetOperation
 - IDMTargetOperation 207
- IDMTargetRenderer
 - IDMTargetRenderer 213
- IDMTBarButtonItem
 - IDMTBarButtonItem 221
- IDMToolBarItem
 - IDMToolBarItem 227
- IDocumentStorage
 - IDocumentStorage 234
- IEmbeddedComponent
 - IEmbeddedComponent 236
- IEmbedderModel
 - IEmbedderModel 244
- IFlatFileStorage
 - IFlatFileStorage 250, 252

- IFont
 - IFont 255
- IG3PointArc
 - IG3PointArc 276
- IGArc
 - IGArc 281
- IGBitmap
 - IGBitmap 289
- IGChord
 - IGChord 303
- IGEllipse
 - IGEllipse 306
- IGLine
 - IGLine 310
- IGList
 - IGList 315
- IGPie
 - IGPie 326
- IGPolygon
 - IGPolygon 331
- IGPolyline
 - IGPolyline 335
- IGrabHandles
 - IGrabHandles 339
- IGraphic
 - IGraphic 351
- IGraphicBundle
 - IGraphicBundle 358
- IGraphicContext
 - IGraphicContext 382
- IGRectangle
 - IGRectangle 407
- IGRegion
 - IGRegion 411
- IGString
 - IGString 425
- IGUIBundle
 - IGUIBundle 434
- image
 - IDMItem 77
- imageFormat
 - IGBitmap 294
- imageOffset
 - IDMItem 78
- imageStyle
 - IDMSourceOperation 154
- IMM24FramesPerSecondTime
 - IMM24FramesPerSecondTime 439
- IMM25FramesPerSecondTime
 - IMM25FramesPerSecondTime 442
- IMM30FramesPerSecondTime
 - IMM30FramesPerSecondTime 445
- IMMAmpMixer
 - IMMAmpMixer 448
- IMMAudioBuffer
 - IMMAudioBuffer 462
- IMMAudioCD
 - IMMAudioCD 471, 479
- IMMAudioCDContents
 - IMMAudioCDContents 484, 488
- IMMCDXA
 - IMMCDXA 493, 496
- IMMConfigurableAudio
 - IMMConfigurableAudio 506
- IMMCuePointEvent
 - IMMCuePointEvent 509
- IMMDevice
 - IMMDevice 533
- IMMDeviceEvent
 - IMMDeviceEvent 547
- IMMDeviceHandler
 - IMMDeviceHandler 550
- IMMDeviceNotifyHandler
 - IMMDeviceNotifyHandler 554
- IMMDigitalVideo
 - IMMDigitalVideo 559, 575
- IMMErrorInfo
 - IMMErrorInfo 580
- IMMFileMedia
 - IMMFileMedia 589
- IMMHourMinSecFrameTime
 - IMMHourMinSecFrameTime 593
- IMMHourMinSecTime
 - IMMHourMinSecTime 598
- IMMMasterAudio
 - IMMMasterAudio 601
- IMMMillisecondTime
 - IMMMillisecondTime 606
- IMMMinSecFrameTime
 - IMMMinSecFrameTime 610
- IMMNotifyEvent
 - IMMNotifyEvent 614
- IMMPassDeviceEvent
 - IMMPassDeviceEvent 617
- IMMPlayableDevice
 - IMMPlayableDevice 628
- IMMPlayerPanel
 - IMMPlayerPanel 633
- IMMPlayerPanelHandler
 - IMMPlayerPanelHandler 639

- IMMPositionChangeEvent
 - IMMPositionChangeEvent 643
- IMMRecordable
 - IMMRecordable 654
- IMMRemovableMedia
 - IMMRemovableMedia 661
- IMMRemovableMediaHandler
 - IMMRemovableMediaHandler 665
- IMMRemovableMediaNotifyHandler
 - IMMRemovableMediaNotifyHandler 669
- IMMSequencer
 - IMMSequencer 672, 675
- IMMSpeed
 - IMMSpeed 677
- IMMTime
 - IMMTime 681
- IMMTrackMinSecFrameTime
 - IMMTrackMinSecFrameTime 690
- IMMWaveAudio
 - IMMWaveAudio 698, 700
- IModel
 - IModel 707
- informSourceOfCompletion
 - IDMTargetRenderer 215
- informTargetOfCompletion
 - IDMSourceRenderer 169
- initialize
 - IComponent 10
 - IComponentStationery 18
 - IEmbeddedComponent 239
 - IView 724
- inl files, description xvii
- insertPoint
 - IGPolyline 336
- instanceFor
 - IDMTargetOperation 210
- intermediatePoint
 - IG3PointArc 277
- internalLeading
 - IFont 260
- invalidate
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 322
 - IMMAudioCDContents::Cursor 490
- invert
 - IGBitmap 297
- inWindow
 - IEmbeddedComponent 242
 - IMGrabbable 438
- IRegionHandle
 - IRegionHandle 709
- isAboveFirst
 - IDMTargetEnterEvent 188
- isAccumulatingBoundingRect
 - IGraphicContext 381
- isAcquired
 - IMMDevice 529
- isAudioEnabled
 - IMMDevice 514
- isAutoPlayEnabled
 - IMMAudioCD 469
- isAvailable
 - IMMErrorInfo 581
- isBitmap
 - IFont 263
- isBitmapOnly
 - IFont 257
- isBold
 - IFont 263
- isClippingRectSet
 - IGString 424
- isCloseOnDestroy
 - IMMDevice 526
- isConnectionSupported
 - IMMDevice 522
- isConnectorEnabled
 - IMMDevice 523
- isContainer
 - IComponent 9
 - IComponentStationery 14
 - IDMItem 78
- isContainerControl
 - IDMHandler 56
- isContinuousPlayEnabled
 - IMMAudioCD 469
- isDragAfter
 - IDMTargetEnterEvent 188
- isDragSource
 - IView 723
- isDropTarget
 - IView 723
- isEmbedded
 - IComponent 9
 - IComponentStationery 14
- isEmpty
 - IGList 317
- isFixed
 - IFont 263

- isGainingUse
 - IMMPassDeviceEvent 618
- isGroup
 - IDMItem 78
- isHitSelectable
 - IGraphic 346
- isHitSelected
 - IGraphic 346
- isHitTesting
 - IGraphicContext 399
- isIGList
 - IGList 317
- isInitialized
 - IEmbeddedComponent 238
- isInPlaceActive
 - IEmbeddedComponent 238
- isItalic
 - IFont 263
- isMediaPresent
 - IMMRemovableMedia 658
- isMonitoringEnabled
 - IMMAmpMixer 455
 - IMMDigitalVideo 567
- isNonPropOnly
 - IFont 258
- isOnRemovableMedia
 - IDMItem 79
- isOpen
 - IDMItem 79
 - IMMDevice 526
- isOpenStringValid
 - IMMAmpMixer 456
 - IMMAudioCD 480
 - IMMCDXA 496
 - IMMDevice 535
 - IMMDigitalVideo 576
 - IMMSequencer 675
 - IMMWaveAudio 701
- isOutline
 - IFont 263
- isPlayingForward
 - IMMDigitalVideo 560
- isReference
 - IDMItem 79
- isServer
 - IComponentStationery 14
- isStrikeout
 - IFont 264
- isStructuredStorage
 - IComponent 10
- isStyle
 - IDMTargetOperation 208
- isTargetTheSource
 - IDMItem 89
- IStructuredStorage
 - IStructuredStorage 712, 713
- isUnderscore
 - IFont 264
- isValid
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 323
 - IMMAudioCDContents 487
 - IMMAudioCDContents::Cursor 490
 - IMMTime 683
- isVectorOnly
 - IFont 258
- isWriteable
 - IMMFileMedia 585
- item
 - IDMOperation 123
- itemCapability
 - IMMDevice 535
- itemStatus
 - IMMDevice 536
- ITransformMatrix
 - ITransformMatrix 716
- IView
 - IView 727

L

- lastError
 - IMMDevice 534
- lastGraphic
 - IGList 317
- length
 - IMMAudioBuffer 462
 - IMMPlayableDevice 621
- lib files, description xvii
- libbmui.a xix
- libbmuis.a xix
- link
 - IDMOperation 130
- linkable
 - IDMItem 99
- load
 - IMMFileMedia 585
- loadBitmap
 - IGBitmap 295

- loadFile
 - IDocumentStorage 231
 - IFlatFileStorage 251
 - IStructuredStorage 712
- loadOnThread
 - IMMFileMedia 586
- lockDoor
 - IMMRemovableMedia 658
- logfont
 - IFont 255

M

- mappingMode
 - IGraphicContext 401
- matchingRMFs
 - IDMItem 92
- maxAscender
 - IFont 260
- maxCharHeight
 - IFont 260
- maxDescender
 - IFont 260
- maximumSpeed
 - IMMDigitalVideo 564
- maximumWindows
 - IMMDigitalVideo 569
- maxLowercaseAscender
 - IFont 260
- maxLowercaseDescender
 - IFont 261
- maxSize
 - IFont 261
- maxUppercaseSize
 - IFont 261
- mediaLoaded
 - IMMRemovableMediaHandler 666
- mediaLoadedId
 - IMMRemovableMedia 663
- mediaType
 - IMMAudioBuffer 464
- mediaUnloaded
 - IMMRemovableMediaHandler 666
- member function names, conventions xix
- menuItem
 - IDM 25
- mergeCopy
 - IGBitmap 297
- mergePaint
 - IGBitmap 297

- messageBox
 - IComponentStationery 17
- IMGrabbable 437
- microphone
 - IMMDevice 540
- minimumSpeed
 - IMMDigitalVideo 564
- minTextWidth
 - IFont 261
- minutes
 - IMMHourMinSecFrameTime 593
 - IMMHourMinSecTime 597
 - IMMMinSecFrameTime 609
 - IMMTime 681
 - IMMTrackMinSecFrameTime 689
- mixerControlValues
 - IMMDevice 532
- mixMode
 - IGraphicBundle 361
 - IGraphicContext 393
- IMM24FramesPerSecondTime 439
- IMM25FramesPerSecondTime 442
- IMM30FramesPerSecondTime 445
- IMMAmpMixer 448
- IMMAudioBuffer 459
- IMMAudioCD 467
- IMMAudioCDContents 484
- IMMAudioCDContents::Cursor 489
- IMMCDXA 493
- IMMConfigurableAudio 498
- IMMCuePointEvent 509
- IMMDevice 512
- IMMDeviceEvent 547
- IMMDeviceHandler 550
- IMMDeviceNotifyHandler 554
- IMMDigitalVideo 557
- IMMErrorInfo 579
- IMMFileMedia 584
- IMMHourMinSecFrameTime 592
- IMMHourMinSecTime 597
- IMMMasterAudio 601
- IMMMillisecondTime 606
- IMMMinSecFrameTime 609
- IMMNotifyEvent 613
- IMMPassDeviceEvent 617
- IMMPlayableDevice 620
- IMMPlayerPanel 631
- IMMPlayerPanelHandler 638
- IMMPositionChangeEvent 643

- IMMRecordable 646
- IMMRemovableMedia 657
- IMMRemovableMediaHandler 665
- IMMRemovableMediaNotifyHandler 669
- IMMSequencer 672
- IMMSpeed 677
- IMMTime 680
- IMMTrackMinSecFrameTime 689
- IMMWaveAudio 694
- mode
 - IMMDevice 525
- model 703
 - IComponent 9
 - IView 723
- monitor
 - IMMDevice 541
- monitorHandle
 - IMMDigitalVideo 567
- mouseDown
 - IGrabHandles 341
- move
 - IDMOperation 130
- moveable
 - IDMItem 100
- moveBy
 - IGRegion 421
- moveSizeTo
 - IView 724
- moveTo
 - IGBitmap 287
 - IGrabHandles 340
 - IGString 427
- Multimedia Classes xvi
 - overview xvi

N

- name
 - IEmbeddedComponent 238
 - IFont 263
- nativeRF
 - IDMItem 92
- nativeRM
 - IDMItem 92
- nativeRMF
 - IDMItem 92
- none
 - IDMItem 99
- normal
 - IGBitmap 297

- normalSpeed
 - IMMDigitalVideo 565
- noSourceRendering
 - IDMSourcePrepareEvent 162
- noStyle
 - IDMImage 68
- notificationHandler
 - IMMDevice 535
 - IMMDeviceNotifyHandler 555
 - IMMRemovableMedia 662
- notifier
 - IComponent 9
 - IModel 704
- notifyOfChange
 - IModel 704
- notSourceErase
 - IGBitmap 297
- number
 - IEmbeddedComponent 238
- numberOfEntries
 - IMMAudioCDContents 485
- numberOfGraphics
 - IGList 317
- numberOfItems
 - IDMOperation 123
- numberOfPoints
 - IDMImage 64
 - IGPolyline 336
- numberOfRenderers
 - IDMHandler 57
 - IDMSourceHandler 147
 - IDMTargetHandler 197
- numberOfTracks
 - IMMAudioCD 469
 - IMMAudioCDContents 487
- numerations, conventions xix

O

- object
 - IDMEFItem 43
 - IDMItem 83
 - IDMMLItem 117
 - IDMSourceBeginEvent 137
 - IDMTargetDropEvent 180
 - IDMTargetEnterEvent 189
- objectArea
 - IEmbeddedComponent 239
- objectMenuBar
 - IGUIBundle 435

- objectToolBar
 - IGUIBundle 435
- objectView
 - IGUIBundle 435
- offset
 - IDMSourceBeginEvent 137
- okToDiscardFile
 - IGUIBundle 436
- open
 - IDMItem 99
 - IMMAmpMixer 449, 456
 - IMMDevice 526, 538
 - IMMFileMedia 587, 589
- openDoor
 - IMMRemovableMedia 659
- openOnThread
 - IMMAmpMixer 450, 457
 - IMMDevice 527, 538
 - IMMFileMedia 587, 590
- openStatus
 - IMMDevice 539
- operation
 - IDMOperation 124
 - IDMSourceOperation 156
 - IDMTargetHelpEvent 203
- operationFrom
 - IDMSourceOperation 157
- operator -
 - IMMTime 684
 - IMMTrackMinSecFrameTime 692
- operator --
 - IMMAudioCDContents::Cursor 491
- operator -=
 - IGRegion 418
 - IMMTime 684
 - IMMTrackMinSecFrameTime 692
- operator !=
 - IG3PointArc 275
 - IGArc 280
 - IGEllipse 305
 - IGLine 309
 - IGPie 326
 - IGPolygon 331
 - IGPolyline 334
 - IGraphic 351
 - IGraphicBundle 358
 - IGRectangle 407
 - IGString 424
 - IMMTime 683
 - ITransformMatrix 715
- operator &=
 - IGRegion 415
- operator +
 - IMMTime 683
 - IMMTrackMinSecFrameTime 692
- operator ++
 - IMMAudioCDContents::Cursor 490
- operator +=
 - IGRegion 417
 - IMMTime 683
 - IMMTrackMinSecFrameTime 692
- operator <
 - IMMTime 684
- operator <=<=
 - IEmbeddedComponent 241
 - IEmbedderModel 247
 - IModel 706
- operator <=
 - IMMTime 684
- operator =
 - IDMImage 64
 - IDMItem 76
 - IFont 256
 - IG3PointArc 276
 - IGArc 281
 - IGEllipse 307
 - IGLine 310
 - IGPie 327
 - IGPolygon 332
 - IGPolyline 335
 - IGraphic 352
 - IGraphicBundle 359
 - IGRectangle 407
 - IGRegion 412
 - IGString 425
 - IMMAudioBuffer 462
 - IMMAudioCDContents 485
 - IMMAudioCDContents::Cursor 490
 - IMMTime 682
 - IMMTrackMinSecFrameTime 691
 - ITransformMatrix 717
- operator ==
 - IG3PointArc 276
 - IGArc 281
 - IGEllipse 306
 - IGLine 310
 - IGPie 326
 - IGPolygon 331
 - IGPolyline 335
 - IGraphic 351

- operator == (*continued*)
 - IGraphicBundle 358
 - IGRectangle 407
 - IGString 424
 - IMMTime 684
 - ITransformMatrix 715
- operator >
 - IMMTime 685
- operator >=
 - IMMTime 685
- operator >>=
 - IEmbeddedComponent 241
 - IEmbedderModel 248
 - IModel 706
- operator ^=
 - IGRegion 419
- operator const char *
 - IMMErrorInfo 581
- operator const IRegionHandle
 - IGRegion 412
- operator INotifier &
 - IModel 705
- operator unsigned long
 - IMM24FramesPerSecondTime 440
 - IMM25FramesPerSecondTime 443
 - IMM30FramesPerSecondTime 446
 - IMMHourMinSecFrameTime 594
 - IMMHourMinSecTime 598
 - IMMMillisecondTime 607
 - IMMMinSecFrameTime 610
 - IMMTime 682
 - IMMTrackMinSecFrameTime 691
- operator Value
 - IRegionHandle 709
- ordinal
 - IMMHourMinSecFrameTime 595
 - IMMHourMinSecTime 599
 - IMMTime 685
- orphan
 - IEmbedderModel 245
- orphanModel
 - IComponent 8
- other
 - IMMDevice 541
- overlay
 - IMMDevice 541
- owner
 - IMMAudioCDContents 487

P

- pageSize
 - IGraphicContext 388
- palette
 - IGBitmap 287
- passDevice
 - IMMDeviceHandler 551
 - IMMRemovableMediaHandler 666
- passDeviceId
 - IMMDevice 542
- paste
 - IMMRecordable 649
- pasteFromBuffer
 - IMMWaveAudio 696
- pasteToBuffer
 - IMMWaveAudio 697
- patternCopy
 - IGBitmap 297
- patternInvert
 - IGBitmap 298
- patternOrigin
 - IGraphicBundle 362
 - IGraphicContext 393
- patternPaint
 - IGBitmap 298
- pause
 - IMMAudioCD 473
 - IMMPlayableDevice 623
 - IMMPlayerPanelHandler 640
- pauseButton
 - IMMPlayerPanel 633
- penColor
 - IGraphicBundle 356
 - IGraphicContext 393
- penEndingStyle
 - IGraphicBundle 367
 - IGraphicContext 393
- penJoiningStyle
 - IGraphicBundle 369
 - IGraphicContext 394
- penPattern
 - IGraphicBundle 364
 - IGraphicContext 394
- penType
 - IGraphicBundle 365
 - IGraphicContext 394
- penWidth
 - IGraphicBundle 366
 - IGraphicContext 394

- pitch
 - IMMAmpMixer 451
- plainText
 - IDM 25
- play
 - IMMAudioCD 474
 - IMMDigitalVideo 560
 - IMMPlayableDevice 624
 - IMMPlayerPanelHandler 640
- playableDevice
 - IMMPlayerPanel 635
- playAt
 - IMMDigitalVideo 561
- playButton
 - IMMPlayerPanel 633
- playDigital
 - IMMDigitalVideo 576
- playFast
 - IMMDigitalVideo 562
- playScan
 - IMMDigitalVideo 562
- playSlow
 - IMMDigitalVideo 563
- point
 - IGPolyline 336
- pointArray
 - IDMImage 64
 - IGPolyline 336
- pointer
 - IDMImage 65
- pointerOffset
 - IDMImage 65
 - IDMSourceOperation 154
- pointSize
 - IFont 263
- pointSizeAt
 - IFont 257
- PointSizeCursor
 - IFont::PointSizeCursor 272
- polygon
 - IDMImage 68
- position
 - IDMOperation 126
 - IDMSourceBeginEvent 137
 - IDMTargetEnterEvent 189
 - IGBitmap 287
 - IGString 427
 - IMMCuePointEvent 510
 - IMMPlayableDevice 622
 - IMMPositionChangeEvent 644
 - positionChange
 - IMMDeviceHandler 552
 - positionChangeId
 - IMMDevice 543
 - positionRelativeTo
 - IDMTargetOperation 209
 - positionTimerId
 - IMMAudioCD 482
 - prepare
 - IDMItem 99
 - prepareAtSource
 - IDMTargetRenderer 217
 - prerollTime
 - IMMDevice 528
 - prerollType
 - IMMDevice 528
 - Presentation Manager messages
 - presSpace
 - IDMTargetEvent 193
 - printDestination
 - IDMSourcePrintEvent 165
 - profile
 - IMMAudioCD 477
 - provideEnterSupport
 - IDMItemProvider 103
 - provideHelpFor
 - IDMItemProvider 104
 - provideLeaveSupport
 - IDMItemProvider 105
 - provideSourceItems
 - IDMItemProvider 105
 - IDMItemProviderFor 108
 - provideTargetItemFor
 - IDMItemProvider 105
 - IDMItemProviderFor 108
 - ptr
 - IDMImage 68

R

- recoordinationHeight
 - IGraphicContext 402
- record
 - IMMDigitalVideo 567
 - IMMRecordable 650
- rect
 - IEmbeddedComponent 242
 - IMGrabbable 438
- redo
 - IMMRecordable 651

- reference
 - IDMItem 99
- reflectHorizontally
 - IGBitmap 292
- reflectVertically
 - IGBitmap 292
- refresh
 - IMMDigitalVideo 569
- IRegionHandle 709
- rejectInComingCalls
 - IComponentStationery 18
- release
 - IMMDevice 529
- releasePresSpace
 - IDMTargetEvent 193
- removableMedia
 - IDMItem 99
- removeAll
 - IGList 318
- removeAllWithId
 - IGList 318
- removeAt
 - IGList 318
- removeAtPosition
 - IGList 318
- removeCuePoint
 - IMMPlayableDevice 621
- removeEntryAt
 - IMMAudioCDContents 485
- removeFirst
 - IGList 318
- removeFont
 - IGString 426
- removeGraphicBundle
 - IGraphic 345
- removeItem
 - IDMOperation 123
- removeLast
 - IGList 318
- removePoint
 - IGPolyline 336
- removeRenderer
 - IDMHandler 57
 - IDMSourceHandler 147
 - IDMTargetHandler 197
- removeRMF
 - IDMItem 92
- removeType
 - IDMItem 88
- renderAtSource (*continued*)
 - IDMTargetRenderer 218
- renderComplete
 - IDMTargetRenderer 218
- renderer
 - IDMHandler 58
 - IDMItem 96
 - IDMSourceHandler 147
 - IDMTargetHandler 197
- renderingFailed
 - IDMTargetEndEvent 185
- renderToName
 - IDMTargetRenderer 219
- replaceAt
 - IGList 314
- replaceEntryAt
 - IMMAudioCDContents 486
- replaceItem
 - IDMOperation 124
- replaceRenderer
 - IDMHandler 58
 - IDMSourceHandler 148
 - IDMTargetHandler 198
- requiresFiles
 - IMMDevice 516
- requiresPreparation
 - IDMItem 79
- resetBackgroundColor
 - IGraphicBundle 357
- resetBackgroundMixMode
 - IGraphicBundle 355
- resetClippingRect
 - IGString 424
- resetDrawOperation
 - IGraphicBundle 359
- resetFillColor
 - IGraphicBundle 357
- resetFillPattern
 - IGraphicBundle 364
- resetFont
 - IGraphicContext 398
- resetMixMode
 - IGraphicBundle 361
- resetPatternOrigin
 - IGraphicBundle 362
- resetPenColor
 - IGraphicBundle 357
- resetPenEndingStyle
 - IGraphicBundle 368
- resetPenJoiningStyle

- resetPenJoiningStyle (*continued*)
 - IGraphicBundle 369
- resetPenPattern
 - IGraphicBundle 364
- resetPenType
 - IGraphicBundle 366
- resetPenWidth
 - IGraphicBundle 366
- resetTransformMatrix
 - IGraphic 348
- resetTransparentColor
 - IGBitmap 288
- resize
 - IEmbeddedComponent 240
- resume
 - IMMAudioCD 474
 - IMMPlayableDevice 623
- reversePoints
 - IGPolyline 336
- rewind
 - IMMPlayerPanelHandler 641
- rewindButton
 - IMMPlayerPanel 633
- rfAny
 - IDM 26
- rfForThisProcess
 - IDMItem 93
- rfFrom
 - IDMItem 93
- rfObject
 - IDM 26
- rfProcess
 - IDM 26
- rfSharedMem
 - IDM 26
- rfText
 - IDM 26
- rfUnknown
 - IDM 26
- rmAny
 - IDM 27
- rmDiscard
 - IDM 27
- rmfFrom
 - IDMItem 93
- rmFile
 - IDM 27
- rmFrom
 - IDMItem 93

- rmfs
 - IDMItem 94
- rmfsFrom
 - IDMItem 94
- rmLibrary
 - IDM 27
- rmObject
 - IDM 27
- rmPrint
 - IDM 28
- rotateBy
 - IGraphic 349
 - ITransformMatrix 719
- rotateBy180
 - IGBitmap 292
- rotateBy270
 - IGBitmap 292
- rotateBy90
 - IGBitmap 293
- rounding
 - IGRectangle 408
- run
 - IComponentStationery 18

S

- sample directory location xxii
- samplesPerSecond
 - IMMAudioBuffer 460
 - IMMConfigurableAudio 500
- save
 - IMMRecordable 651
- saveAs
 - IMMRecordable 652
- saveFile
 - IDocumentStorage 232
 - IFlatFileStorage 251
 - IStructuredStorage 712
- saveHeadphonesSetting
 - IMMMasterAudio 602
- saveSpeakersSetting
 - IMMMasterAudio 603
- saveVolume
 - IMMMasterAudio 603
- scaleBy
 - IGraphic 349
 - ITransformMatrix 720
- seconds
 - IMMHourMinSecFrameTime 593
 - IMMHourMinSecTime 597

seconds (*continued*)

- IMMMinSecFrameTime 609
- IMMTime 681
- IMMTrackMinSecFrameTime 690

seek

- IMMPlayableDevice 626

seekToEnd

- IMMPlayableDevice 626

seekToStart

- IMMPlayableDevice 626

selectComponent

- IEmbedderModel 246

selectedComponent

- IEmbedderModel 247

selectedFormat

- IDMSourceRenderEvent 176

selectedMechanism

- IDMSourceRenderEvent 176

selectedRMF

- IDMItem 96

sendCommand

- IMMDevice 536

sequencer

- IMMDevice 541

serialNumber

- IMMAudioCDContents 487

setAllEmphasis

- IFont 264

setAlternateWindowHandle

- IDMSourcePrepareEvent 161

setArea

- IEmbeddedComponent 241

setAspect

- IEmbeddedComponent 237

setBackgroundColor

- IGraphicBundle 357
- IGraphicContext 394

setBackgroundMixMode

- IGraphicBundle 355
- IGraphicContext 395

setBalance

- IMMAmpMixer 451

setBass

- IMMAmpMixer 452

setBitmap

- IDMImage 65

setBitsPerSample

- IMMAudioBuffer 460
- IMMConfigurableAudio 501

setBlockAlignment

- IMMAudioBuffer 461
- IMMConfigurableAudio 501

setBlockAlignment (*continued*)

- IMMAudioBuffer 461
- IMMConfigurableAudio 501

setBold

- IFont 264

setBytesPerSecond

- IMMAudioBuffer 461
- IMMConfigurableAudio 502

setChanged

- IModel 705

setChannels

- IMMAudioBuffer 461
- IMMConfigurableAudio 502

setCharHeight

- IFont 266

setCharSize

- IFont 266

setCharWidth

- IFont 266

setClippingRect

- IGString 424

setClipRegion

- IGraphicContext 403

setCloseOnDestroy

- IMMDevice 527

setCompletion

- IDMSourceRenderEvent 176

setContainer

- IDMItem 79

setContainerName

- IDMItem 80

setContainerObject

- IDMOperation 121

setContainerRefreshOff

- IDMOperation 121

setContainerRefreshOn

- IDMOperation 122

setContents

- IDMItem 83

setContentType

- IMMAudioBuffer 463

setCurrentDrawingPosition

- IGraphicContext 383

setData

- IMMAudioBuffer 462

setDebugSupport

- IDMOperation 122

setDefaultBackgroundColor

- IGraphicContext 386

setDefaultBackgroundMixMode

setDefaultBackgroundMixMode (continued)	
IGraphicContext	386
setDefaultDrawOperation	
IGraphicContext	386
setDefaultFillColor	
IGraphicContext	386
setDefaultFillPattern	
IGraphicContext	386
setDefaultMixMode	
IGraphicContext	387
setDefaultOperation	
IDMTargetEnterEvent	190
setDefaultPatternOrigin	
IGraphicContext	387
setDefaultPenColor	
IGraphicContext	387
setDefaultPenEndingStyle	
IGraphicContext	387
setDefaultPenJoiningStyle	
IGraphicContext	387
setDefaultPenPattern	
IGraphicContext	387
setDefaultPenType	
IGraphicContext	388
setDefaultPenWidth	
IGraphicContext	388
setDefaultSourceHandler	
IDMHandler	50
setDefaultSourceRenderer	
IDMRenderer	133
setDefaultStyle	
IDMImage	67
setDefaultTargetHandler	
IDMHandler	50
setDefaultTargetRenderer	
IDMRenderer	133
setDestination	
IMMDigitalVideo	570
setDirection	
IFont	264
IGArc	282
setDiscTitle	
IMMAudioCD	470
setDocumentName	
IComponent	11
setDragInfo	
IDMOperation	129
setDragResult	
IDMOperation	129
setDrawOperation	
IGraphicBundle	360
IGraphicContext	395
setDropIndicator	
IDMTargetEnterEvent	190
setDropOffset	
IDMTargetOperation	209
setDropPosition	
IDMTargetDropEvent	181
setDropStatus	
IDMItem	77
setElement11	
ITransformMatrix	718
setElement12	
ITransformMatrix	719
setElement21	
ITransformMatrix	719
setElement22	
ITransformMatrix	719
setElement31	
ITransformMatrix	719
setElement32	
ITransformMatrix	719
setEnclosingRect	
IGArc	283
IGEllipse	307
IGPie	327
IGRectangle	409
setEndPoint	
IG3PointArc	277
IGLine	311
setFileName	
IDocumentStorage	233
setFillColor	
IGraphicBundle	357
IGraphicContext	395
setFillMode	
IGPolygon	332
setFillPattern	
IGraphicBundle	364
IGraphicContext	395
setFirstTimeEntered	
IDMTargetOperation	211
setFont	
IGraphicContext	399
IGString	426
setFontAngle	
IFont	266
setFontShear	
IFont	266

- setFormat
 - IMMAudioBuffer 463
 - IMMConfigurableAudio 503
- setGain
 - IMMAmpMixer 452
- setGraphicBundle
 - IGraphic 345
 - IGraphicContext 396
- setGroup
 - IDMItem 80
- setHitApertureSize
 - IGraphicContext 400
- setHitPoint
 - IGraphicContext 400
- setHitSelectable
 - IGraphic 346
- setHitSelected
 - IGraphic 346
- setId
 - IGraphic 347
- setImage
 - IDMItem 77
- setImageStyle
 - IDMSourceOperation 155
- setIntermediatePoint
 - IG3PointArc 277
- setItalic
 - IFont 265
- setItemProvider
 - IDMHandler 56
- setLastError
 - IMMDevice 534
- setMappingMode
 - IGraphicContext 401
- setMediaType
 - IMMAudioBuffer 464
- setMixerControlValues
 - IMMDevice 532
- setMixMode
 - IGraphicBundle 361
 - IGraphicContext 396
- setMMTime
 - IMMTime 686
- setMonitorWindow
 - IMMDigitalVideo 567
- setName
 - IFont 265
- setNativeRMF
 - IDMItem 94
- setNoSourceRendering
- setNoSourceRendering (*continued*)
 - IDMSourcePrepareEvent 162
- setNotificationHandler
 - IMMDevice 535
 - IMMDeviceNotifyHandler 556
 - IMMRemovableMedia 662
- setNumberOfPoints
 - IDMImage 65
- setObject
 - IDMItem 84
- setObjectArea
 - IEmbeddedComponent 241
- setOnRemovableMedia
 - IDMItem 80
- setOpen
 - IDMItem 80
- setOpenStatus
 - IMMDevice 539
- setOperation
 - IDMOperation 125
- setOutline
 - IFont 265
- setPageSize
 - IGraphicContext 388
- setPalette
 - IGBitmap 287
- setPassDeviceRequested
 - IMMDevice 537
- setPatternOrigin
 - IGraphicBundle 362
 - IGraphicContext 396
- setPenColor
 - IGraphicBundle 357
 - IGraphicContext 396
- setPenEndingStyle
 - IGraphicBundle 368
 - IGraphicContext 397
- setPenJoiningStyle
 - IGraphicBundle 369
 - IGraphicContext 397
- setPenPattern
 - IGraphicBundle 364
 - IGraphicContext 397
- setPenType
 - IGraphicBundle 366
 - IGraphicContext 397
- setPenWidth
 - IGraphicBundle 367
 - IGraphicContext 398
- setPitch

setPitch (<i>continued</i>)	
IMMAmpMixer	452
setPlayableDevice	
IMMPlayerPanel	635
setPoint	
IGPolyline	337
setPointArray	
IDMImage	65
setPointer	
IDMImage	66
setPointerOffset	
IDMImage	66
IDMSourceOperation	154
setPoints	
IGPolyline	337
setPointSize	
IFont	265
setPosition	
IDMOperation	126
setProfile	
IMMAudioCD	477
setProgram	
IMMAudioCD	478
setRecoordinationHeight	
IGraphicContext	402
setRect	
IEmbeddedComponent	242
IMGrabbable	438
setReference	
IDMItem	81
setRenderer	
IDMHandler	58
IDMItem	96
IDMSourceHandler	148
IDMTargetHandler	198
setRequiresPreparation	
IDMItem	81
setRetry	
IDMSourceRenderEvent	176
setRMFs	
IDMItem	95
setRounding	
IGRectangle	408
setSamplesPerSecond	
IMMAudioBuffer	461
IMMConfigurableAudio	503
setSelectedRMF	
IDMItem	96
setSource	
IDMOperation	125
setSourceName	
IDMItem	81
setSourceOperation	
IDMSourceDiscardEvent	140
IDMSourceOperation	157
IDMSourcePrintEvent	165
setSourceWindowHandle	
IDMItem	81
IDMOperation	126
setSpeedFormat	
IMMDevice	530
setStackingPercentage	
IDMSourceOperation	155
setStartAngle	
IGArc	283
IGPie	328
setStartingPoint	
IG3PointArc	277
IGLine	311
setStretchSize	
IDMImage	66
setStrikeout	
IFont	265
setStyle	
IDMImage	66
IDMTargetOperation	209
setSupportedRMFs	
IDMRenderer	134
setSupportedTypes	
IDMRenderer	134
setSweepAngle	
IGArc	283
IGPie	328
setSystemMixerHandle	
IMMDevice	533
setTargetCanRetry	
IDMSourcePrepareEvent	162
setTargetInfo	
IDMSourceRenderEvent	177
IDMTargetDropEvent	181
setTargetName	
IDMItem	82
setTargetOperation	
IDMTargetOperation	211
setTargetWindowHandle	
IDMOperation	126
setText	
IGString	427
setTimeFormat	
IMMDevice	530

- setTimeToOrdinal
 - IMMHourMinSecFrameTime 595
 - IMMHourMinSecTime 599
 - IMMTime 685
- setToFirst
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 323
 - IMMAudioCDContents::Cursor 491
- setToIdentity
 - ITransformMatrix 719
- setToIndex
 - IMMAudioCDContents::Cursor 491
- setToLast
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 323
 - IMMAudioCDContents::Cursor 491
- setToNext
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 323
 - IMMAudioCDContents::Cursor 491
- setToPrevious
 - IFont::FaceNameCursor 270
 - IFont::PointSizeCursor 273
 - IGList::Cursor 323
 - IMMAudioCDContents::Cursor 491
- setTrackTitle
 - IMMAudioCD 470
- setTransformMatrix
 - IGraphic 349
- setTransformMethod
 - IGraphic 349
- setTransparentColor
 - IGBitmap 288
- setTreble
 - IMMAmpMixer 453
- setTrueType
 - IDMItem 89
- setTypes
 - IDMItem 89
- setUnderscore
 - IFont 265
- setUserParameter
 - IMMDevice 537
- setValid
 - IMMTime 686
- setViewClient
 - IView 724
- setViewOption
 - IGBitmap 294
- setViewPortRect
 - IGraphicContext 389
- setVolume
 - IMMAudioCD 472
 - IMMConfigurableAudio 504
 - IMMDevice 515
 - IMMMasterAudio 603
 - IMMSequencer 673
- setWhoDiscards
 - IDMSourceDiscardEvent 140
- setWhoPrints
 - IDMSourcePrintEvent 165
- setWindow
 - IMMDigitalVideo 570
- setWindowFont
 - IFont 267
- setWorldTransformMatrix
 - IGraphicContext 404
- shortName
 - IComponentStationery 14
- showGrabHandles
 - IEmbeddedComponent 240
- size
 - IGBitmap 287
- sizeTo
 - IGBitmap 287
 - IView 724
- slope
 - IGLine 311
- slowSpeed
 - IMMDigitalVideo 565
- sort
 - IGList 318
- source
 - IDMOperation 126
 - IDMSourceBeginEvent 137
- sourceAnd
 - IGBitmap 298
- sourceBegin
 - IDMSourceHandler 149
- sourceContainer
 - IDMCnrItem 38
- sourceDiscard
 - IDMItem 85
 - IDMSourceHandler 149
 - IDMSourceRenderer 169
 - IDMTBarButtonItem 222
- sourceEnd

- sourceEnd (*continued*)
 - IDMEFItem 42
 - IDMItem 86
 - IDMMenuItem 111
 - IDMMLItem 116
 - IDMSourceHandler 150
 - IDMSourceRenderer 169
- sourceErase
 - IGBitmap 298
- sourceInvert
 - IGBitmap 298
- sourceItemFor
 - IDMItem 90
- sourceName
 - IDMItem 82
- sourceOperation
 - IDMItem 96
 - IDMSourceDiscardEvent 140
 - IDMSourceOperation 157
 - IDMSourcePrintEvent 166
- sourcePaint
 - IGBitmap 298
- sourcePrepare
 - IDMItem 86
 - IDMSourceHandler 150
 - IDMSourceRenderer 170
- sourcePrint
 - IDMItem 87
 - IDMSourceHandler 150
 - IDMSourceRenderer 170
- sourceRectangle
 - IMMDigitalVideo 571
- sourceRender
 - IDMItem 87
 - IDMSourceHandler 151
 - IDMSourceRenderer 170
- sourceToolBar
 - IDMTBarButtonItem 223
- sourceToolBarButton
 - IDMTBarButtonItem 223
- sourceWindow
 - IDMItem 82
 - IDMOperation 127
- sourceWindowHandle
 - IDMItem 82
 - IDMOperation 127
- speaker
 - IMMDevice 541
- speed
 - IMMDigitalVideo 565
- speed (*continued*)
 - IMMSpeed 678
- speedFormat
 - IMMDevice 531
- stackingPercentage
 - IDMSourceOperation 155
- Standard Control Classes xvi
 - overview xvi
- startAngle
 - IGArc 283
 - IGPie 328
- startBoundaryAccumulation
 - IGraphicContext 381
- startHitTesting
 - IGraphicContext 400
- startingPoint
 - IG3PointArc 277
 - IGLine 311
- startOfTrack
 - IMMAudioCDContents 487
- startPositionTracking
 - IMMPlayableDevice 622
- startScanningBackward
 - IMMAudioCD 475
- startScanningForward
 - IMMAudioCD 475
- stationery
 - IComponentStationery 16
- stepBackward
 - IMMPlayerPanelHandler 641
- stepBackwardButton
 - IMMPlayerPanel 633
- stepForward
 - IMMPlayerPanelHandler 641
- stepForwardButton
 - IMMPlayerPanel 633
- stepFrame
 - IMMPlayableDevice 627
- stop
 - IMMAudioCD 476
 - IMMPlayableDevice 625
 - IMMPlayerPanelHandler 641
- stopBoundaryAccumulation
 - IGraphicContext 381
- stopButton
 - IMMPlayerPanel 633
- stopHandlingEventsFor
 - IMMDeviceHandler 551
 - IMMRemovableMediaHandler 666

- stopHitTesting
 - IGraphicContext 401
- stopPositionTracking
 - IMMPlayableDevice 623
- storage
 - IComponent 10
- strContents
 - IDMItem 100
- stretch
 - IDMImage 68
- stretchSize
 - IDMImage 67
- IStructuredStorage 711
- style
 - IDMImage 67
 - IDMTargetOperation 209
- subscriptOffset
 - IFont 261
- subscriptSize
 - IFont 261
- successCode
 - IMMNotifyEvent 615
- superscriptOffset
 - IFont 262
- superscriptSize
 - IFont 262
- supportedOperations
 - IDMItem 82
- supportedOperationsFor
 - IDMItem 85
 - IDMMenuItem 112
 - IDMTBarButtonItem 224
- supportedRMFs
 - IDMRenderer 134
- supportedTypes
 - IDMRenderer 134
- supportsAudio
 - IMMDevice 516
- supportsCommand
 - IMMDevice 516
- supportsDigitalTransfer
 - IMMDevice 517
- supportsDisableEject
 - IMMDevice 517
- supportsEject
 - IMMDevice 517
- supportsOperation
 - IDMTargetRenderer 215
- supportsOverlayGraphics
 - IMMDigitalVideo 558
- supportsPlay
 - IMMDevice 518
- supportsRecord
 - IMMDevice 518
- supportsRecordInsertion
 - IMMDevice 519
- supportsReverse
 - IMMDigitalVideo 558
- supportsRMF
 - IDMItem 95
- supportsSave
 - IMMDevice 519
- supportsSizing
 - IMMDigitalVideo 558
- supportsStreaming
 - IMMDevice 519
- supportsStretchToFit
 - IMMDigitalVideo 559
- supportsVideo
 - IMMDevice 520
- supportsVolumeAdjustment
 - IMMDevice 520
- supportsWaveFormat
 - IMMWaveAudio 697
- sweepAngle
 - IGArc 283
 - IGPie 328
- systemMixerHandle
 - IMMDevice 533

T

- targetCanRetry
 - IDMSourcePrepareEvent 162
 - IDMTargetEndEvent 185
- targetDrop
 - IDMCnrlItem 36
 - IDMEFIItem 43
 - IDMItem 87
 - IDMMLItem 117
 - IDMTargetHandler 199
 - IDMTBarButtonItem 221
 - IDMToolBarItem 228
- targetEnd
 - IDMItem 88
 - IDMTargetHandler 199
- targetEnter
 - IDMTargetHandler 200
- targetHelp
 - IDMTargetHandler 200

- targetInfo
 - IDMSourceRenderEvent 177
 - IDMTargetDropEvent 181
 - IDMTargetEndEvent 185
- targetLeave
 - IDMTargetHandler 200
- targetName
 - IDMItem 82
 - IDMSourceRenderEvent 177
- targetOperation
 - IDMItem 97
 - IDMTargetOperation 211
- targetRender
 - IDMTargetRenderer 216
- targetRenderComplete
 - IDMTargetRenderer 216
- targetRenderPrepare
 - IDMTargetRenderer 217
- targetToolBar
 - IDMTBarButtonItem 223
- targetToolBarButton
 - IDMTBarButtonItem 223
- targetWindow
 - IDMOperation 127
 - IDMSourceRenderEvent 177
- targetWindowHandle
 - IDMOperation 127
 - IDMSourceRenderEvent 177
- text
 - IDM 25
 - IGString 427
 - IMMErrorInfo 581
- textLines
 - IFont 262
- textWidth
 - IFont 262
- thousandths
 - IMMTime 681
- throwMMError
 - IMMErrorInfo 581
- timeFormat
 - IMMDevice 531
- tokenForWPSObject
 - IDMItem 90
- toolBarButton
 - IDM 25
- topGraphicUnderPoint
 - IGList 316
- track
 - IMMAudioCDContents 487

- track (*continued*)
 - IMMTrackMinSecFrameTime 690
- trackBackward
 - IMMAudioCD 476
- trackForward
 - IMMAudioCD 477
- trackStartedId
 - IMMAudioCD 482
- trackTitle
 - IMMAudioCD 470
- transformMatrix 715
 - IGraphic 350
- transformMethod
 - IGraphic 350
- translateAudioFlag
 - IMMConfigurableAudio 507
 - IMMDigitalVideo 576
- translateBy
 - IGraphic 350
 - ITransformMatrix 720
- transparent
 - IDMImage 69
- transparentColor
 - IGBitmap 288
- transposeXForY
 - IGBitmap 293
- treble
 - IMMAmpMixer 453
- trueType
 - IDMItem 89
- tryToLoadBitmap
 - IGBitmap 295
- Two-Dimensional Graphic Classes xvi
 - overview xvi
- type names, conventions xix
- types
 - IDMItem 89

U

- undo
 - IMMRecordable 652
- uniqueDiscIdentifier
 - IMMAudioCD 480
- unknown
 - IDM 25
 - IDMItem 100
 - IDMOperation 130
- unlockDoor
 - IMMRemovableMedia 659

- upc
 - IMMAudioCD 471
 - IMMCDXA 494
- useBitmapOnly
 - IFont 258
- useDefaultMonitorWindow
 - IMMDigitalVideo 568
- useDefaultWindow
 - IMMDigitalVideo 571
- useNonPropOnly
 - IFont 258
- userParameter
 - IMMCuePointEvent 510
 - IMMDevice 537
 - IMMNotifyEvent 614
 - IMMPositionChangeEvent 644
- useVectorOnly
 - IFont 258

V

- vertical
 - IGraphicBundle 374
- videoDisc
 - IMMDevice 541
- videoFileHeight
 - IMMDigitalVideo 572
- videoFileName
 - IMMDigitalVideo 572
- videoFileWidth
 - IMMDigitalVideo 572
- videoHeight
 - IMMDigitalVideo 573
- videoTape
 - IMMDevice 541
- videoWidth
 - IMMDigitalVideo 573
- view 722
 - IGUIBundle 435
- viewClient
 - IView 725
- viewOption
 - IGBitmap 294
- viewPortRect
 - IGraphicContext 389
- volume
 - IMMAudioCD 472
 - IMMConfigurableAudio 504
 - IMMDevice 515
 - IMMMasterAudio 604

- volume (*continued*)
 - IMMSequencer 673

W

- wasDragAfter
 - IDMTargetOperation 210
- wasPassDeviceRequested
 - IMMDevice 537
- wasTargetSuccessful
 - IDMSourceEndEvent 143
- waveAudio
 - IMMDevice 541
- white
 - IGBitmap 298
- whoDiscards
 - IDMSourceDiscardEvent 141
- whoPrints
 - IDMSourcePrintEvent 166
- worldTransformMatrix
 - IGraphicContext 404
- writeToFile
 - IGBitmap 294

X

- X/Motif messages
- xxxxxxx.inf xix

Communicating Your Comments to IBM

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume IV

Version 3.5

Publication No. S33H-5042-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
 - United States and Canada: 416-448-6161
 - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
 - Internet: torrcf@vnet.ibm.com
 - IBMLink: [toribm\(torrcf\)](mailto:toribm(torrcf)@vnet.ibm.com)
 - IBM/PROFS: [torolab4\(torrcf\)](mailto:torolab4(torrcf)@vnet.ibm.com)
 - IBMMAIL: [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@vnet.ibm.com)

Readers' Comments — We'd Like to Hear from You

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume IV

Version 3.5

Publication No. S33H-5042-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
S33H-5042-00



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape

Please do not staple

Fold and Tape

S33H-5042-00

Cut or Fold
Along Line

IBM®

Part Number: 33H5042

Printed in U.S.A.

S33H-5042-00



33H5042

