IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume II

Version 3.5

Document Number S33H-5040-00

IBM VisualAge for C++ for Windows

S33H-5040-00

Open Class Library Reference
Volume II

Version 3.5

IBM

IBM VisualAge for C++ for Windows

S33H-5040-00

Open Class Library Reference
Volume II

Version 3.5

Note

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

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:

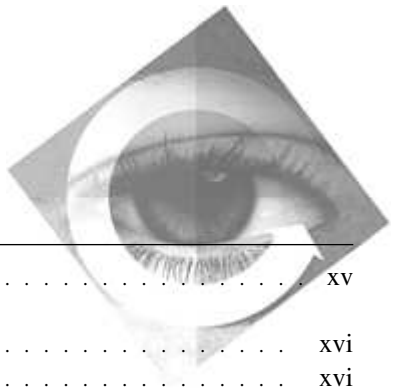
IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada. M3C 1H7

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 xv

Notices xvi
Programming Interface Information xvi
Trademarks xvii

About This Book xviii
Who Should Use This Book xviii
Conventions Used in This Book xviii
How This Reference is Organized xix
About the User Interface Class Library xx
User Interface Class Library Conventions xxiii
File Names xxiii
Class Names and Member Names xxv
Function Return Types and Function Arguments xxv
Using Obsolete or Ignored Member Functions xxvi
A Note about Samples and Examples xxviii

Part 2. Description of Classes 1

Class Hierarchy by Category 2

I3StateCheckBox 8

I3StateCheckBox::Style 16

IAccelerator 17

IAcceleratorKey 22

IAcceleratorTable 30

IAcceleratorTable::Cursor 39

IAccelTblHandle 42

IAnchorBlockHandle 44

IApplication 46

© Copyright IBM Corp. 1992, 1996

IBaseComboBox 52

IBaseComboBox::Cursor 75

IBaseComboBox::Style 79

IBaseListBox 81

IBaseListBox::Cursor 101

IBaseListBox::Style 105

IBaseSpinButton 106

IBaseSpinButton::Style 121

IBidiSettings 122

IBitmapHandle 131

IButton 134

IButton::Style 142

IButtonNotifyHandler 143

ICheckBox 146

ICheckBox::Style 154

IClipboard 155

IClipboard::Cursor 167

IClipboardHandler 170

IColor 174

IComboBox 182

IComboBox::Style 196

IComboBoxNotifyHandler 197

ICommand 200

ICommandConnectionTo 206

ICommandEvent 210

ICommandHandler 214

IContextHandle 219

IControl 221

IControl::Style 226

IControlEvent 227

ICoordinateSystem 230

ICritSec 234

ICurrentApplication 237

ICurrentThread 243

IDeviceColor 253

IDisplayHandle 256

IDrawItemEvent 258

IDynamicLinkLibrary 261

IEditHandler 267

IEntryField 271

IEntryField::Style 297

IEntryFieldNotifyHandler 299

IEnumHandle 302

IEvent 304

© IBM Corp. 1992, 1996

IEventData 313

IEventParameter1 319

IEventParameter2 320

IEventResult 321

IFocusHandler 322

IFrameEvent 326

IFrameExtension 329

IFrameExtensions 338

IFrameFormatEvent 339

IFrameHandler 342

IFrameWindow 349

IFrameWindow::Style 392

IFrameWindowNotifyHandler 393

IGroupBox 396

IGroupBox::Style 404

IGUIColor 405

IHandle 408

IHandler 411

IHelpErrorEvent 416

IHelpHandler 420

IHelpHyperlinkEvent 429

IHelpMenuBarEvent 432

IHelpNotifyEvent 434

IHelpSubitemNotFoundEvent 437

IHelpTutorialEvent 440

IHelpWindow 442

IHelpWindow::Settings 459

IHelpWindow::Style 464

IHighEventParameter 465

IKey 466

IKey::KeyModifier 475

IKeyboardConnectionTo 477

IKeyboardEvent 481

IKeyboardHandler 490

IListBox 495

IListBox::Style 508

IListBoxDrawItemEvent 509

IListBoxDrawItemHandler 513

IListBoxNotifyHandler 518

IListBoxSizeItemEvent 521

ILowEventParameter 524

IMenu 525

IMenu::Cursor 550

IMenu::Style 553

© IBM Corp. 1992, 1996

IMenuBar 554

IMenuBar::Style 561

IMenuDrawItemEvent 562

IMenuDrawItemHandler 565

IMenuDrawItemHandler::DrawFlag 570

IMenuEvent 571

IMenuHandle 574

IMenuHandler 576

IMenuItem 582

IMenuItem::Attribute 599

IMenuItem::Style 600

IMenuNotifyHandler 601

IMessageBox 604

IMessageBox::Style 613

IMessageQueueHandle 615

IModuleHandle 618

IMouseClickEvent 620

IMouseConnectionTo 624

IMouseEvent 628

IMouseHandler 631

IMousePointerEvent 635

IMousePointerHandler 638

IMultiLineEdit 641

IMultiLineEdit::Style 668

IMultiLineEditNotifyHandler 669

INumericSpinButton 672

INumericSpinButton::Style 682

INumericSpinButtonNotifyHandler 683

IObjectWindow 686

IOutlineBox 689

IOutlineBox::Style 697

IPaintConnectionTo 698

IPaintEvent 701

IPaintHandler 706

IPointerHandle 709

IPopUpMenu 712

IPresSpaceHandle 717

IPrivateResource 719

IProcedureAddress 723

IProcessId 727

IProfile 729

IProfile::Cursor 738

IProfileHandle 741

IProgressIndicator 743

© IBM Corp. 1992, 1996

IProgressIndicator::Style 770

IProgressIndicatorNotifyHandler 771

IPushButton 774

IPushButton::Style 785

IRadioButton 786

IRadioButton::Style 795

IRecoordHandler 796

IResizeEvent 800

IResizeHandler 802

IResource 805

IResourceId 808

IResourceLibrary 811

IResourceLock 820

IScrollBar 823

IScrollBar::Style 838

IScrollBarNotifyHandler 839

IScrollEvent 842

IScrollHandler 845

ISelectHandler 851

ISemaphoreHandle 855

ISettingButton 857

ISettingButtonNotifyHandler 863

ISharedResource 866

IShowListHandler 870

ISlider 874

ISlider::Style 887

ISliderArmHandler 888

ISliderDrawHandler 892

ISpinHandler 896

IStaticText 900

IStaticText::Style 916

IStringHandle 917

ISubmenu 919

ISubmenu::Cursor 931

ISWP 934

ISWPArray 940

ISystemBitmapHandle 942

ISystemMenu 946

ISystemPointerHandle 952

ITextControl 955

ITextControlNotifyHandler 960

ITextSpinButton 963

ITextSpinButton::Cursor 974

ITextSpinButton::Style 977

© IBM Corp. 1992, 1996

ITextSpinButtonNotifyHandler 978

IThread 981

IThread::Cursor 1001

IThreadFn 1004

IThreadHandle 1006

IThreadId 1008

IThreadMemberFn 1010

ITimer 1013

ITimer::Cursor 1019

ITimerFn 1021

ITimerMemberFn 1023

ITimerMemberFn0 1026

ITitle 1029

ITitleNotifyHandler 1041

IWindow 1044

IWindow::ChildCursor 1100

IWindow::ExceptionFn 1103

IWindow::Style 1105

IWindowHandle 1107

IWindowNotifyHandler 1111

Glossary 1114

Bibliography 1131

Index 1132

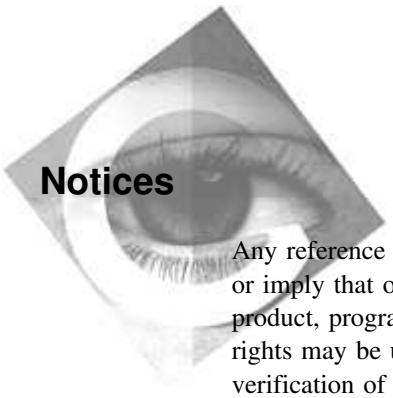
© IBM Corp. 1992, 1996

Part 1. About this Book

Notices xvi
Programming Interface Information xvi
Trademarks xvii

About This Book xviii
Who Should Use This Book xviii
Conventions Used in This Book xviii
How This Reference is Organized xix
About the User Interface Class Library xx
User Interface Class Library Conventions xxiii
File Names xxiii
Class Names and Member Names xxv
Function Return Types and Function Arguments xxv
Using Obsolete or Ignored Member Functions xxvi
A Note about Samples and Examples xxviii

© Copyright IBM Corp. 1992, 1996



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++.

© Copyright IBM Corp. 1992, 1996

Trademarks

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

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

Windows is a trademark of Microsoft Corporation.

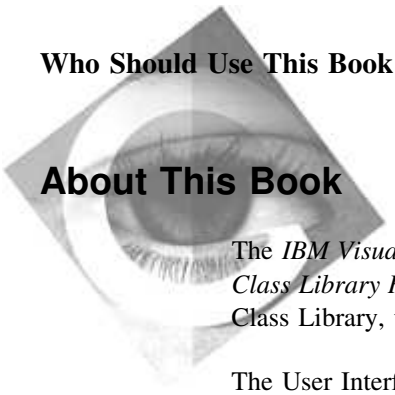
PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

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..

© IBM Corp. 1992, 1996



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 xix 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.

© Copyright IBM Corp. 1992, 1996

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 xx 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

© IBM Corp. 1992, 1996

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();
```

Win	PM	Motif
N	*Y*	*N*

2
```
void
  initialize( unsigned long windowId,
              const IWindowHandle& parent,
              const IWindowHandle& owner,
              unsigned long style,
              const IRectangle& initial);
```

Win	PM	Motif
Y	*Y*	*N*

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

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.

© IBM Corp. 1992, 1996

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
I*xxxxxxx*.CPP	Source code
I*xxxxxxx*.H	Constant definitions file
I*xxxxxxx*.HPP	Class interface file
I*xxxxxxx*.INL	Inline functions

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

File Name and Extension	Description
i*xxxxxxx*.cpp	Source code
i*xxxxxxx*.hpp	Class interface file
i*xxxxxxx*.h	Constant definitions file
i*xxxxxxx*.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.

© IBM Corp. 1992, 1996

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, files 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.

CPPWOU3I.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
libibmui.a	Static object library for User Interface Class Library	/usr/lpp/xlC/lib
libibmuis.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
  ```

© IBM Corp. 1992, 1996

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

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 wrappered 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.

© IBM Corp. 1992, 1996

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 thes 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:

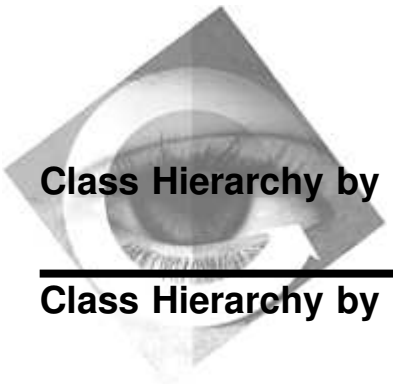
```
\ibmcppw\samples\ioc
```

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

© Copyright IBM Corp. 1992, 1996



Class Hierarchy by Category

Class Hierarchy by Category

© Copyright IBM Corp. 1992, 1996

Application Control Classes

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

```
IBase
 ├─ICritSec
 ├─IHandle
 │  ├─IContextHandle
 │  ├─IProcessId
 │  ├─IProfileHandle
 │  ├─ISemaphoreHandle
 │  └─IThreadId
 ├─IProcedureAddress
 ├─IResourceId
 └─IVBase
    ├─IApplication
    │  └─ICurrentApplication
    ├─IClipboard
    ├─IClipboard::Cursor
    ├─IHandler
    │  └─IClipboardHandler
    ├─IProfile
    ├─IProfile::Cursor
    ├─IRefCounted
    │  ├─IThreadFn
    │  │  └─IThreadMemberFn
    │  └─ITimerFn
    │     ├─ITimerMemberFn
    │     └─ITimerMemberFn0
    ├─IResource
    │  ├─IPrivateResource
    │  └─ISharedResource
    ├─IResourceLibrary
    │  └─IDynamicLinkLibrary
    ├─IResourceLock
    ├─IThread
    │  └─ICurrentThread
    ├─IThread::Cursor
    ├─ITimer
    └─ITimer::Cursor
```

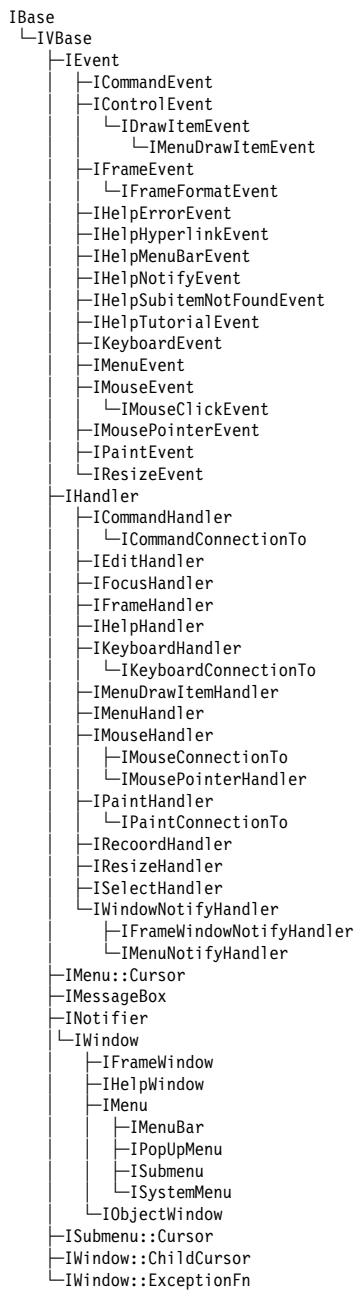
© IBM Corp. 1992, 1996

Base Window, Menu, Handler, and Event Classes

Provide support for the basic windows, handlers, events, and menus used by the applications you develop.

```
ISequence
 └─IFrameExtensions
IBase
 ├─IAccelerator
 ├─IAcceleratorKey
 ├─IAcceleratorTable
 ├─IBitFlag
 │  ├─IFrameWindow::Style
 │  ├─IHelpWindow::Style
 │  ├─IKey::KeyModifier
 │  ├─IMenu::Style
 │  ├─IMenuBar::Style
 │  ├─IMenuDrawItemHandler::DrawFlag
 │  ├─IMenuItem::Attribute
 │  ├─IMenuItem::Style
 │  ├─IMessageBox::Style
 │  └─IWindow::Style
 ├─ICommand
 ├─ICoordinateSystem
 ├─IEventData
 ├─IEventParameter1
 ├─IEventParameter2
 ├─IEventResult
 ├─IFrameExtension
 ├─IHandle
 │  ├─IAccelTblHandle
 │  ├─IAnchorBlockHandle
 │  ├─IBitmapHandle
 │  │  └─ISystemBitmapHandle
 │  ├─IDisplayHandle
 │  ├─IEnumHandle
 │  ├─IMenuHandle
 │  ├─IMessageQueueHandle
 │  ├─IModuleHandle
 │  ├─IPointerHandle
 │  │  └─ISystemPointerHandle
 │  ├─IPresSpaceHandle
 │  ├─IStringHandle
 │  └─IWindowHandle
 ├─IHelpWindow::Settings
 ├─IHighEventParameter
 ├─IKey
 ├─ILowEventParameter
 ├─IMenuItem
 ├─ISWP
 ├─ISWPArray
 └─IVBase
    ├─IAcceleratorTable::Cursor
    ├─IBidiSettings
    └─IColor
       ├─IDeviceColor
       └─IGUIColor
```

Base Window, Menu, Handler, and Event Classes ...



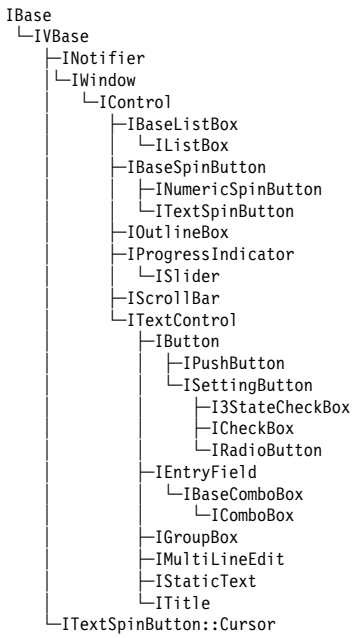
© IBM Corp. 1992, 1996

Standard Control Classes

Provide support for the basic controls, such as entry fields, static text, and buttons used by the applications you develop.

```
IBase
 │ IBitFlag
 │  ├─I3StateCheckBox::Style
 │  ├─IBaseComboBox::Style
 │  ├─IBaseListBox::Style
 │  ├─IBaseSpinButton::Style
 │  ├─IButton::Style
 │  ├─ICheckBox::Style
 │  ├─IControl::Style
 │  ├─IEntryField::Style
 │  ├─IGroupBox::Style
 │  ├─IListBox::Style
 │  ├─IMultiLineEdit::Style
 │  ├─INumericSpinButton::Style
 │  ├─IOutlineBox::Style
 │  ├─IProgressIndicator::Style
 │  ├─IPushButton::Style
 │  ├─IRadioButton::Style
 │  ├─IScrollBar::Style
 │  ├─ISlider::Style
 │  ├─IStaticText::Style
 │  └─ITextSpinButton::Style
 └─IVBase
    ├─IBaseComboBox::Cursor
    ├─IBaseListBox::Cursor
    ├─IEvent
    │  ├─IControlEvent
    │  │  ├─IDrawItemEvent
    │  │  │  └─IListBoxDrawItemEvent
    │  │  └─IListBoxSizeItemEvent
    │  └─IScrollEvent
    └─IHandler
       ├─IListBoxDrawItemHandler
       ├─IScrollHandler
       ├─IShowListHandler
       ├─ISliderArmHandler
       ├─ISliderDrawHandler
       ├─ISpinHandler
       └─IWindowNotifyHandler
          ├─IListBoxNotifyHandler
          ├─INumericSpinButtonNotifyHandler
          ├─IProgressIndicatorNotifyHandler
          ├─IScrollBarNotifyHandler
          ├─ITextControlNotifyHandler
          │  ├─IButtonNotifyHandler
          │  │  └─ISettingButtonNotifyHandler
          │  ├─IEntryFieldNotifyHandler
          │  │  └─IComboBoxNotifyHandler
          │  ├─IMultiLineEditNotifyHandler
          │  └─ITitleNotifyHandler
          └─ITextSpinButtonNotifyHandler
```

Standard Control Classes ...



© IBM Corp. 1992, 1996

I3StateCheckBox

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
ISettingButton
I3StateCheckBox

Inherited By None.

Header File i3statbx.hpp

Members

Member	Page	Member	Page
Constructor	9	enableAutoSelect	9
autoSelect	14	isAutoSelect	9
calcMinimumSize	13	isHalftone	11
classDefaultStyle	14	selectHalftone	11
convertToGUIStyle	11	setDefaultStyle	12
defaultStyle	11	~I3StateCheckBox	10
disableAutoSelect	9		

The I3StateCheckBox class creates and manages three-state check-box control windows. A three-state check box is a square box with associated text that represents a choice. Unlike a regular check box, which has only two states (selected and deselected), a three-state check box has three states:

selected
The check box is filled to indicate that the item is selected.

indeterminate
The check box is displayed as a halftone to indicate that the choice is indeterminate.

deselected
The check box is cleared to indicate that the item is deselected.

You can process the selection of a three-state check box by deriving from the ISelectHandler class and adding the handler to either the three-state check box or its owner window. See ISelectHandler (p. 851) for information about that class.

© Copyright IBM Corp. 1992, 1996

Public Functions

Auto Select

Use auto select members to query and modify the autoSelect style of a three-state check box object. The autoSelect style determines whether the state of the three-state check box is automatically changed when the user clicks on it.

disableAutoSelect

Removes the autoSelect (p. 14) style from the three-state check box control. If auto select is disabled, the application is responsible for changing the state of the three-state check box when the check box is clicked.

```
virtual I3StateCheckBox&
  disableAutoSelect();
```

Win	PM	Motif
Y	*Y*	*N*

enableAutoSelect

Sets the three-state check box control to the autoSelect (p. 14) style. If auto select is enabled, the state of the check box is automatically cycled between the selected, indeterminate, and deselected states when the check box is clicked.

```
virtual I3StateCheckBox&
  enableAutoSelect( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*N*

isAutoSelect If the three-state check box control has the autoSelect (p. 14) style set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isAutoSelect() const;
```

Win	PM	Motif
Y	*Y*	*N*

Constructors

You can construct and destruct objects of the I3StateCheckBox class. You cannot copy or assign I3StateCheckBox objects because both the copy constructor and assignment operator are private functions.

I3StateCheckBox

© IBM Corp. 1992, 1996

I3StateCheckBox

1 `I3StateCheckBox( unsigned long id,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial = IRectangle ( ),`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	*Y*	*N*

Constructs a three-state check box control and an object for it using the following parameters:

id The window ID of the three-state check box.

parent The parent window.

owner The owner window.

initial The initial position and size of the three-state check box you are constructing. The default is IRectangle (Vol. I). Optional.

style The three-state check box's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2 `I3StateCheckBox( unsigned long id,`
`IWindow* parent );`

Win	PM	Motif
Y	*Y*	*N*

Constructs an object from a dialog (parent) window, and the ID of a three-state check box control on that dialog window.

id The window ID of the three-state check box.
parent The dialog (parent) window.

3 `I3StateCheckBox( const IWindowHandle& handle );`

Win	PM	Motif
Y	*Y*	*N*

Constructs the object using the handle of an existing three-state check box window.

handle The window handle of an existing three-state check box control.

~I3StateCheckBox

```
virtual
  ~I3StateCheckBox();
```

Win	PM	Motif
Y	*Y*	*N*

Selection

Use these members to query and set the halftone selection state of a three-state check box. The halftone selection state refers to the state of the three-state check box when it is neither selected nor not selected. This state is sometimes referred to as indeterminate. The selection state of a button is typically changed by the user clicking on it using the mouse or pressing a key to select it. You can also change or query the state using User Interface Class Library functions.

isHalftone

Returns true if the three-state check box control has been selected as halftone.

```
Boolean
  isHalftone() const;
```

Win	PM	Motif
Y	Y	N

selectHalftone

Sets a three-state check box control to the halftone selection state.

```
I3StateCheckBox&
  selectHalftone();
```

Win	PM	Motif
Y	Y	N

Styles

These style members provide a set of valid styles for this class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the current default style. This is the same as classDefaultStyle unless setDefaultStyle has been called. See classDefaultStyle (p. 14) for information about classDefaultStyle, and see setDefaultStyle (p. 12) for information about that function.

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

I3StateCheckBox

setDefaultStyle

Sets the default style for all subsequent three-state check boxes.

style Use the styles provided by I3StateCheckBox Styles (p. 13) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

ISettingButton		
deselect	enableAutoSelect	isAutoSelect
disableAutoSelect	enableNotification	isSelected

IButton		
allowsMouseClickFocus	disableMouseClickFocus	highlight
backgroundColor	enableMouseClickFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size that this three-state check box control should be, based on the text string length and the current font.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Functions

ISettingButton		
passEventToOwner		

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid styles for this class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

© IBM Corp. 1992, 1996

I3StateCheckBox

autoSelect The auto select style determines whether the state of the three-state check box is automatically changed when the user clicks on it. If auto select is enabled, the state of the check box automatically cycles between the selected, indeterminate, and deselected states. If auto select is disabled, the application must change the state of the three-state check box when the check box is clicked.

```
static const Style
  autoSelect;
```

Win	PM	Motif
Y	*Y*	*N*

classDefaultStyle

Provides the original default style for this class, which is the following: I3StateCheckBox::autoSelect | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Data

ISettingButton		
selectId		

IButton		
buttonClickId	**noPointerFocus**	

ITextControl		
textId		

IControl		
group	**tabStop**	

IWindow		
activeColorId	**disabledBackgroundColorId**	**noStyle**
backgroundColorId	**disabledForegroundColorId**	**positionId**
borderColorId	**enableId**	**saveBits**
clipChildren	**focusId**	**shadowColorId**
clipSiblings	**fontId**	**sizeId**

IWindow		
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Data

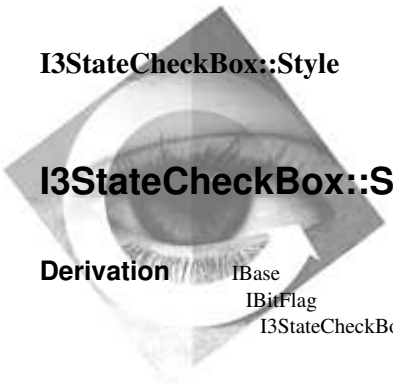
IBase		
recoverable	unrecoverable	

Nested Classes

I3StateCheckBox contains the following nested classes:

I3StateCheckBox::Style (see page 16)

© IBM Corp. 1992, 1996



I3StateCheckBox::Style

Derivation

IBase
IBitFlag
I3StateCheckBox::Style

Inherited By None.

Header File i3statbx.hpp

The nested class I3StateCheckBox::Style provides a set of valid styles for the I3StateCheckBox (p. 8) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IAccelerator

Derivation

IBase
  IAccelerator

Inherited By

None.

Header File

iaccel.hpp

Members

Member	Page	Member	Page
Constructor	20	remove	18
handle	18	reset	18
isSet	18	set	19
owner	21	~IAccelerator	20

The IAccelerator class adds accelerators to a window's menu system. *Accelerators* are a list of shortcut keys and associated command IDs that are stored in resource files. You can activate these shortcut tables so that when you press the shortcut keys, you trigger the associated command actions. Typically, the command actions are the same as certain menu choices.

PM Use this class to associate shortcut keys for a single frame window or an entire application. The shortcut keys are in effect only when a window associated with the accelerator table has the keyboard focus.

Motif In AIX, the User Interface Class Library does not create or maintain IAccelTblHandle (p. 42) objects. Motif has no equivalent concept.

The User Interface Class Library activates accelerators by installing each shortcut key listed in the accelerator resource on a menu item whose identifier matches the accelerator command value. If no match is found, the shortcut key is not installed and, thus, does not activate any command actions when it is pressed. As a result, you must construct the menu system for the window before you can successfully install shortcut keys for the window. If a menu is reset, the shortcut keys must also be set again. Once you have installed shortcut keys, they cannot be removed.

You must set the *owner* parameter of the supported constructors with the frame window containing the menu system to which the accelerators are to be added.

The Motif XmNaccelerator and XmNacceleratorText resources are set for each CascadeButton and ToggleButton gadget that is assigned a shortcut key.

© Copyright IBM Corp. 1992, 1996

Additionally in AIX, you cannot do the following:

- Define shortcut keys that run system commands
- Use these member functions:
  - IAccelerator::handle, which always returns 0
  - IAccelerator::isSet
  - IAccelerator::remove
  - IAccelerator::reset

Public Functions

Accelerator Tables

Accelerator tables contain a list of shortcut key definitions. You can set, remove, and query the accelerator table currently in use.

handle Returns the lAccelTblHandle (p. 42) of the accelerator in effect.

```
IAccelTblHandle
  handle() const;
```

Win	PM	Motif
Y	*Y*	*I*

isSet If the accelerator is in effect, true is returned.

```
Boolean
  isSet() const;
```

Win	PM	Motif
Y	*Y*	*I*

remove Removes the accelerator without restoring the one that was previously in effect. Calling this function destroys the underlying accelerator table. If you then try to access the same IAccelTblHandle (p. 42) after removing it, an exception is thrown.

```
IAccelerator&
  remove();
```

Win	PM	Motif
Y	*Y*	*I*

Exceptions	
IAccessError	The accelerator table could not be removed. The accelerator table or the window to which it was applied may be invalid.

reset Removes the accelerator and restores the one that was previously in effect.

```
IAccelerator&
  reset();
```

Win	PM	Motif
Y	*Y*	*I*

set Changes to the new accelerator table. The accelerator table previously in use is saved.

Note: Only one previous accelerator table is saved.

haccel Accelerator handle.

accelResId Accelerator resource ID.

1
```
IAccelerator&
  set( const IResourceId& accelResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want to specify the resource library that the User Interface Class Library uses to load the accelerator table.

PM Saves the current accelerator and changes to the new accelerator.

2
```
IAccelerator&
  set( const IAccelTblHandle& haccel );
```

Win	PM	Motif
Y	Y	I

Use this function when you already have the accelerator table loaded.

PM Saves the current accelerator and changes to the new accelerator.

Exceptions	
IAccessError	The accelerator table could not be associated with the window or application. Possibly *haccel* has an invalid value.

3
```
IAccelerator&
  set( unsigned long accelResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want the User Interface Class Library to load the accelerator table from the default resource library.

PM Saves the current accelerator and changes to the new accelerator.

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IAccelerator objects because both the copy constructor and assignment operator are private functions.

The constructors accept an optional parameter, *owner*, for the window that uses the accelerator keys. If you specify an owner, the accelerators are only in effect when the specified IWindow object has the focus. If you do not specify *owner* on the constructor, IAccelerator applies to all windows in the application that share the current message queue, but it has no effect on other applications that are currently running.

© IBM Corp. 1992, 1996

IAccelerator

Note: When an application destructs the IAccelerator object, the IAccelerator object destroys the underlying accelerator table. Therefore, the following does *not* work:

```
IAccelerator* pac1 = new IAccelerator(myAccelRes);
IAccelTblHandle haccel1 = pac1->accel();
delete pac1;
pac1 = new IAccelerator(haccel1);
        // The accelerator handle (haccel1)
        // is no longer valid.
```

IAccelerator

1

```
IAccelerator( const IAccelTblHandle& haccel = 0,
              IWindow* owner = 0 );
```

Win	PM	Motif
Y	*Y*	*N*

Creates an object from an IAccelTblHandle (p. 42) and an optional window to apply the table to. Use this constructor when you already have the accelerator table loaded.

2

```
IAccelerator( const IResourceId& accelResId,
              IWindow* owner = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Creates an object from the IResourceId (p. 808) of the accelerator table and an optional window to apply the table to. Use this constructor when you want to load the accelerator table from the resource library of your choice.

Motif You must specify a nonzero value for *owner*.

3

```
IAccelerator( unsigned long accelResId,
              IWindow* owner = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Creates an object from the IResourceId (p. 808) of the accelerator table in the default resource library and an optional window to apply the table to.

Motif You must specify a nonzero value for *owner*.

~IAccelerator

The object and the underlying accelerator table are destroyed. If a window was associated with the accelerator and that window is still valid, its original accelerator table is restored. If, after restoring the original accelerator table, there is a saved accelerator table that is not in use, that table is also destroyed.

```
~IAccelerator();
```

Win	PM	Motif
Y	*Y*	*Y*

Exceptions	
IAccessError	The accelerator table or the saved previous accelerator table is invalid.

Window

You can apply accelerator keys to either a single window or to all windows in your application. Only when a frame window with accelerators is active (one of its child windows has the input focus) will your pressing an accelerator key cause the command associated with the accelerator key to be run.

owner Returns the IWindow (p. 1044) to which the accelerator applies. If the accelerator table is applicable to all windows sharing the current message queue, 0 is returned.

```
IWindow*
  owner() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IAcceleratorKey

Derivation IBase
IAcceleratorKey

Inherited By None.

Header File iaccelky.hpp

Members

Member	Page	Member	Page
Constructor	25	operator ==	24
actionType	23	setCommand	23
asACCEL	27	setHelpKey	23
character	27	setKey	28
commandId	23	setSystemCommand	24
isLike	24	uniqueKeyFor	29
keyModifier	27	virtualKey	28
operator !=	24	~IAcceleratorKey	27
operator =	27		

The IAcceleratorKey class represents a shortcut key for running an action. Such a key is called a *keyboard accelerator* or *accelerator key*. An accelerator key is defined as a keystroke and the action that the keystroke runs. An example of a keyboard accelerator is Alt+F4. Pressing this key combination typically runs the Close system command, which closes the active frame window.

The keystroke consists of a *character key* or *virtual key*, and an optional *modifier*. A character key has the character it types on its key top, such as **A** or **a**. A virtual key, such as Enter or F5. generally does not have an associated character value. Along with a character key or virtual key, you can optionally require the user to press a modifier, which can be the Shift, Alt, or Ctrl key, or a combination of these keys.

The action that an accelerator key runs can be an application command, system command, or a help request. While a standard use for an accelerator key is to run the action defined for a menu item, an accelerator key does not require a corresponding menu item. Defining an accelerator key does not cause the text of a menu item that invokes the same action to be modified in any way. You are responsible for changing the text of a menu item if you want it to identify an accelerator key (for example, by changing the text in a menu resource or by calling IMenuItem::setText (p. 588)).

© Copyright IBM Corp. 1992, 1996

You use IAcceleratorKey objects to build, query, and modify the contents of an IAcceleratorTable (p. 30) object.

Public Functions

Action Definitions

An accelerator key associates an action with a keystroke. The action can be an application command, system command, or help request.

actionType Identifies if the accelerator key runs an application command, system command, or invokes help. You can use this function to determine whether to treat the value returned by IAcceleratorKey::commandId (p. 23) as the identifier of an application or system command.

```
ICommand::ActionType
  actionType() const;
```

Win	PM	Motif
Y	*Y*	*N*

commandId If the accelerator key runs an application or system command, this function returns the identifier of the command. If the accelerator key invokes help, this function returns 0. You can determine the type of action that the key runs using IAcceleratorKey::actionType (p. 23).

```
ICommand::CommandId
  commandId() const;
```

Win	PM	Motif
Y	*Y*	*N*

setCommand Defines the accelerator key to run the specified application command as its action.

```
IAcceleratorKey&
  setCommand( ICommand::CommandId commandId );
```

Win	PM	Motif
Y	*Y*	*N*

commandId

The identifier for the application command that the accelerator key runs. You can specify any command identifier for this parameter. ICommand (p. 201) provides a list of identifiers for common application commands, although you must provide the implementation for these commands (in a command handler (p. 214), for example).

Win Limit *commandId* to values under 0xF000 to avoid conflicts with system command identifiers. If you use a system command identifier, the resulting action will be a system command instead of an application command.

setHelpKey Defines the accelerator key to generate a contextual help request as its action.

© IBM Corp. 1992, 1996

```
IAcceleratorKey&
  setHelpKey();
```

Win	PM	Motif
Y	Y	N

setSystemCommand

Defines the accelerator key to run the specified system command as its action.

```
IAcceleratorKey&
  setSystemCommand( ICommand::CommandId commandId );
```

Win	PM	Motif
Y	Y	N

commandId

The identifier for the system command that the accelerator key runs. You can specify the identifier for any system command as the value of this parameter. ICommand (p. 201) provides a list of identifiers for portable system commands, however.

Win The value of *commandId* must be a system command provided by the Windows operating system. If you use an application command identifier, the resulting action will be an application command instead of a system command.

Comparisons

Comparison functions indicate if two IAcceleratorKey objects are the same, or if they define actions for the same keystroke combination.

isLike

Identifies if two accelerator keys use the same key combination. This function returns true if both use the same character key or virtual key, and the same combination of Alt, Ctrl, and Shift. This function ignores any differences in the actions that the two accelerator keys run.

```
Boolean
  isLike( const IAcceleratorKey& key ) const;
```

Win	PM	Motif
Y	Y	N

operator !=

Specifies if two accelerator keys use a different key combination or run a different action.

```
Boolean
  operator !=( const IAcceleratorKey& key ) const;
```

Win	PM	Motif
Y	Y	N

PM This function also accounts for the AF_SCANCODE and AF_LONEKEY flags that IAcceleratorKey does not externalize. These flags are kept for IAcceleratorKey objects constructed from an _ACCEL structure or queried from an IAcceleratorTable (p. 30) object that was constructed from a Presentation Manager accelerator table.

operator ==

Specifies if two accelerator keys use the same key combination and run the same action.

```
Boolean
  operator ==( const IAcceleratorKey& key ) const;
```

Win	PM	Motif
Y	Y	N

PM This function also accounts for the AF_SCANCODE and AF_LONEKEY flags that IAcceleratorKey does not externalize. These flags are kept for IAcceleratorKey objects constructed from an _ACCEL structure or queried from an IAcceleratorTable (p. 30) object that was constructed from a Presentation Manager accelerator table.

Constructors

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

IAcceleratorKey

1
```
IAcceleratorKey( const IAcceleratorKey& key );
```

Win	PM	Motif
Y	Y	N

Use this copy constructor to create an accelerator key object from an existing IAcceleratorKey object. The key combination and action used by the new object are copied from the existing object.

2
```
IAcceleratorKey(
   IKey::VirtualKey virtualKey,
   const IKey::KeyModifier& modifier,
   ICommand::ActionType actionType =
     ICommand::applicationCommand,
   ICommand::CommandId commandId = 0 );
```

Win	PM	Motif
Y	Y	N

Use this constructor to create an accelerator key for a virtual key.

virtualKey The virtual key for the accelerator key. You can specify any virtual key code provided by the presentation system for this parameter, although IKey (p. 467) provides a list of portable virtual keys.

modifier A combination of modifier or auxiliary keys that the user must press along with the virtual key to run the action. IKey (p. 467) defines the set of available modifiers. Specifying a value of IKey::noModifier (p. 467) requires the user to press the virtual key by itself to run the accelerator key's action.

actionType The type of action that the accelerator key runs.

commandId
The identifier for the application or system command that the accelerator key runs. This value is ignored if *actionType* has a value of ICommand::help (p. 205).

© IBM Corp. 1992, 1996

IAcceleratorKey

Exceptions	
IInvalidParameter	You specified an uninitialized ICommand::ActionType value.

3 `IAcceleratorKey(`
`  const IString& characterKey,`
`  const IKey::KeyModifier& modifier,`
`  ICommand::ActionType actionType =`
`  ICommand::applicationCommand,`
`  ICommand::CommandId commandId = 0 );`

Win	PM	Motif
Y	*Y*	*N*

Use this constructor to create an accelerator key for a character key.

characterKey
The character key for the accelerator key. The first character of *characterKey* is used to define the accelerator key. Any other characters in the string are ignored.

modifier A combination of modifier or auxiliary keys that the user must press along with the character key to run the action. IKey (p. 467) defines the set of available modifiers. Specifying a value of IKey::noModifier (p. 467) requires the user to press the character key by itself to run the accelerator key's action.

actionType The type of action that the accelerator key runs.

commandId
The identifier for the application or system command that the accelerator key runs. This value is ignored if *actionType* has a value of ICommand::help.

Exceptions	
IInvalidParameter	You specified an uninitialized ICommand::ActionType value.

4 `IAcceleratorKey( const ACCEL& accelerator );`

Win	PM	Motif
Y	*Y*	*N*

Use this constructor to create an object from a data structure defined by the presentation system.

PM *accelerator* is a reference to an _ACCEL structure.

Win *accelerator* is a reference to a tagACCEL structure.

operator = Use this assignment operator to set the value of an existing accelerator key object equal to that of another IAcceleratorKey object. This function copies both the key combination and action.

```
IAcceleratorKey&
  operator =( const IAcceleratorKey& key );
```

Win	PM	Motif
Y	Y	N

~IAcceleratorKey

```
~IAcceleratorKey();
```

Win	PM	Motif
Y	Y	N

Conversions

Conversion functions provide an alternative way to represent accelerator keys.

asACCEL Converts an accelerator key object into a data structure defined by the presentation system for use with its APIs.

```
ACCEL
  asACCEL() const;
```

Win	PM	Motif
Y	Y	N

PM The accelerator key is converted into an _ACCEL structure.

Win The accelerator key is converted into a tagACCEL structure.

Note: Some information may be lost during this conversion, because a tagACCEL structure has no way to differentiate the different types of actions supported by IAcceleratorKey.

Key Definitions

An accelerator key associates an action with a key combination. The keystroke can be a character or virtual key that is optionally combined with the Alt, Ctrl, and/or Shift keys.

character Returns the character key defined for the accelerator key. If the accelerator key uses a virtual key instead of a character key, this function returns a 0-length IString.

```
IString
  character() const;
```

Win	PM	Motif
Y	Y	N

keyModifier Returns the modifier or auxiliary keys defined for the accelerator key. The user must press these keys with the character or virtual key to run the action of the accelerator key. The returned value will be IKey::noModifier (p. 467) or some

© IBM Corp. 1992, 1996

combination of the IKey::alt (p. 467), IKey::ctrl (p. 467), and IKey::shift (p. 467) values.

```
IKey::KeyModifier
  keyModifier() const;
```

Win	PM	Motif
Y	Y	N

setKey

Assigns the specified key combination to the accelerator key. The keystroke consists of a character or virtual key, and an optional modifier key, such as Alt, Ctrl, Shift, or a combination of any of these keys.

1

```
IAcceleratorKey&
  setKey(
   const IString& accelKey,
   const IKey::KeyModifier& modifier = IKey::noModifier );
```

Win	PM	Motif
Y	Y	N

Use this function to define a character key combination for the accelerator key. The first character held by *accelKey* is used as the character key. Any other characters in the string are ignored.

2

```
IAcceleratorKey&
  setKey(
   IKey::VirtualKey virtualKey,
   const IKey::KeyModifier& modifier = IKey::noModifier );
```

Win	PM	Motif
Y	Y	N

Use this function to assign a virtual keystroke to the accelerator key.

Note: To make Alt, Ctrl, or Shift an accelerator key, specify a virtual key value of IKey::kAlt (p. 467), IKey::kCtrl (p. 468), or IKey::kShift (p. 473) without a modifier (rather than specify IKey::kNoKey (p. 472) for *accelKey* with a modifier of IKey::alt (p. 467), IKey::ctrl (p. 467), or IKey::shift (p. 467)).

Exceptions	
IInvalidParameter	You cannot specify IAccelerator::kNoKey as the virtual key.

virtualKey

Returns the virtual key defined for this accelerator key.

If the accelerator key uses a platform-dependent virtual key, this function returns a value other than one of those provided by IKey (p. 467).

If the accelerator key uses a character key instead of a virtual key, this function returns IKey::kNoKey (p. 472). You can also use the function IAcceleratorKey::character (p. 27) to determine if the accelerator key uses a character key.

```
IKey::VirtualKey
  virtualKey() const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Comparisons

Comparison functions indicate if two IAcceleratorKey objects are the same, or if they define actions for the same keystroke combination.

uniqueKeyFor

Returns a value that uniquely identifies the key combination used by the accelerator key. IAcceleratorTable (p. 30) uses this value to place IAcceleratorKey objects into a keyed set.

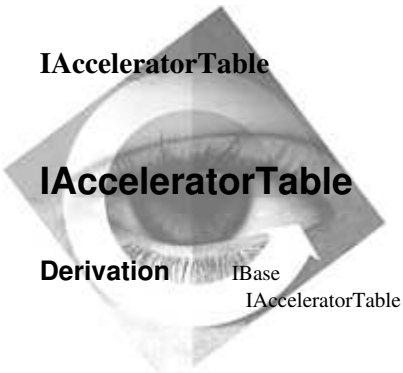
```
static unsigned long
  uniqueKeyFor( const IAcceleratorKey& key );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IAcceleratorTable

Derivation

IBase
IAcceleratorTable

Inherited By None.

Header File iacceltb.hpp

Members

Member	Page	Member	Page
Constructor	34	locateKeyLike	37
addKey	31	operator =	36
addOrReplaceKey	32	removeAllKeys	32
containsKeyLike	37	removeKeyAt	32
destroyHandle	36	removeKeyLike	33
handle	37	replaceKeyAt	33
keyAt	37	~IAcceleratorTable	36
keyCount	37		

The IAcceleratorTable class represents a set of *accelerator keys* (also known as *keyboard accelerators*), which are shortcut keys for running an action. These keys are represented by IAcceleratorKey (p. 22) objects.

Any window can have an accelerator table associated with it. You can assign an accelerator table to a window using the IAcceleratorTable class. You can also set the accelerator table for a window by doing any of the following:

- Constructing an IFrameWindow (p. 349) object with the style IFrameWindow::accelerator (p. 383)
- Using the function IWindow::setAcceleratorHandle (p. 1048)
- Using the IAccelerator (p. 17) class

However, for your application to add, replace, remove, or query the keys in the accelerator table, you must use the IAcceleratorTable and IAcceleratorKey classes.

If you press an accelerator key used by the window with the input focus, that window is sent the resulting command or help request. If the window does not translate the key into a command or help request, a window in the parent window chain (up to and including the first frame window) could translate the key. In this case, the parent window that has the key in its accelerator table receives a command event, or the window with the input focus receives a help request. An accelerator key does not generate a keyboard event.

© Copyright IBM Corp. 1992, 1996

Note: The presentation system defines system accelerator keys that it will automatically enable based on the system menu items used by a frame window. You do not need to create an accelerator table to use the system accelerator keys, and these accelerator keys will not be reflected in the IAcceleratorTable object you create for a window.

Public Functions

Changing Keys

IAcceleratorTable allows you to modify a collection of accelerator keys by adding, replacing, or removing individual keys.

If you create an IAcceleratorTable object for a window, using the constructor that accepts a const IWindow* parameter or calling IWindow::acceleratorTable (p. 1047), you can cause any changes you make to the accelerator keys to be immediately reflected in the accelerator key processing of the window. Each of the functions that change a key has an *updateWindow* argument that allows you to control whether the contents of the accelerator table should be updated to the window.

If you defer updating the window, you can later call IWindow::setAcceleratorTable (p. 1048) to update the accelerator keys used by the window.

addKey Adds the specified accelerator key to the table. If the accelerator key was added, this function returns true. Otherwise the table remains unchanged, and the function returns false. The accelerator key will not be added if an existing entry in the table has the same keystroke (the same character or virtual key and the same Alt, Ctrl, Shift modifiers) as the key you are trying to add.

If the accelerator key was added, all other cursors for this accelerator table become undefined.

Win	PM	Motif
Y	*Y*	*N*

```
Boolean
  addKey( const IAcceleratorKey& newKey,
          Boolean updateWindow = true );
```

newKey The accelerator key being added to the table.

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

© IBM Corp. 1992, 1996

IAcceleratorTable

addOrReplaceKey

Adds the specified accelerator key to the table, or replaces an existing entry. If the accelerator table does not contain an entry with the same keystroke (the same character or virtual key and the same Alt, Ctrl, Shift modifiers), this function adds the specified key and returns true. Otherwise it replaces the matching entry and returns false.

If the accelerator key was added, all other cursors for this accelerator table become undefined.

```
Boolean
  addOrReplaceKey( const IAcceleratorKey& newKey,
                   Boolean updateWindow = true );
```

Win	PM	Motif
Y	*Y*	*N*

newKey The new accelerator key for the table.

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

removeAllKeys

Empties the accelerator table so it contains no keys. Following this function, the accelerator table is still valid and can still be assigned to a window.

Following use of this function, all cursors for this accelerator table become undefined.

```
IAcceleratorTable&
  removeAllKeys( Boolean updateWindow = true );
```

Win	PM	Motif
Y	*Y*	*N*

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

removeKeyAt

Removes the accelerator key at the cursor from the table.

All other cursors for this accelerator table become undefined.

```
IAcceleratorTable&
  removeKeyAt( Cursor& cursor,
               Boolean updateWindow = true );
```

Win	PM	Motif
Y	Y	N

cursor Cursor that points at the key to remove.

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

removeKeyLike

Removes from the table the accelerator key entry with the same keystroke (the same character or virtual key and the same Alt, Ctrl, Shift modifiers) as the specified key. If an entry is removed, this function returns true. Otherwise the table remain unchanged, and the function returns false.

If an accelerator key was removed, all cursors for this accelerator table become undefined.

```
Boolean
  removeKeyLike( const IAcceleratorKey& key,
                 Boolean updateWindow = true );
```

Win	PM	Motif
Y	Y	N

key Key that contains the key combination and action to match.

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

replaceKeyAt

Replaces the accelerator key at the cursor with the specified accelerator key.

Following use of this function, all other cursors for this accelerator table become undefined.

```
IAcceleratorTable&
  replaceKeyAt( const Cursor& cursor,
                const IAcceleratorKey& newKey,
                Boolean updateWindow = true );
```

Win	PM	Motif
Y	Y	N

cursor Cursor that points at the key to replace.

© IBM Corp. 1992, 1996

newKey Replacement key.

updateWindow
Flag that specifies if the window that the IAcceleratorTable object was created for should be updated with the current contents of the accelerator table. This flag is ignored unless the IAcceleratorTable object was constructed with a nonzero const IWindow* parameter, or was returned by the IWindow::acceleratorTable (p. 1047) function.

Constructors

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

IAcceleratorTable

1 `IAcceleratorTable( const IAccelTblHandle& accelTblHandle );`

Win	PM	Motif
Y	*Y*	*N*

Use this constructor to create an IAcceleratorTable object that contains the accelerator keys in an existing accelerator table.

accelTblHandle
The handle of the existing accelerator table.

Note: Changing the contents of the IAcceleratorTable object does not alter the accelerator keys used by the existing accelerator table.

W32s Creating an instance of IAcceleratorTable with this constructor throws an exception.

Exceptions	
IInvalidRequest	Constructing an object of this class is not supported on the Win32s platform.

2 `IAcceleratorTable( Boolean systemQueueKeys );`

Win	PM	Motif
N	*Y*	*N*

Use this constructor to create an IAcceleratorTable object that contains the accelerator keys used by the system queue of the presentation system, or an IAcceleratorTable object that contains no accelerator keys.

systemQueueKeys
Flag that specifies how the accelerator table should be constructed. If false, the accelerator table is created empty. Otherwise, the accelerator table contains the system accelerator keys.

Note: Changing the contents of the IAcceleratorTable object does not alter the system accelerator keys.

3 `IAcceleratorTable( const IWindow* window = 0 );`

Win	PM	Motif
Y	*Y*	*N*

Use this constructor to create an IAcceleratorTable object that contains the accelerator keys used by the specified window. Passing a window to this constructor is equivalent to calling IWindow::acceleratorTable (p. 1047).

window If 0, the accelerator table is constructed with no keys. Otherwise, the accelerator table contains the accelerator keys of the window.

Note: You can cause changes to the contents of the accelerator table to be immediately reflected by the window. See the functions that modify an accelerator table (p. 31) for more information.

W32s Creating an instance of IAcceleratorTable with this constructor throws an exception.

Exceptions	
IInvalidRequest	Constructing an object of this class is not supported on the Win32s platform.

4 `IAcceleratorTable( const IResourceId& resId );`

Win	PM	Motif
Y	*Y*	*N*

Use this constructor to create an IAcceleratorTable object that contains accelerator keys loaded from an accelerator table resource in a resource library.

resId Identifies the accelerator table and resource library.

W32s Creating an instance of IAcceleratorTable with this constructor throws an exception.

Exceptions	
IInvalidRequest	Constructing an object of this class is not supported on the Win32s platform.

5 `IAcceleratorTable( const IAcceleratorTable& accelTbl );`

Win	PM	Motif
Y	*Y*	*N*

Use this copy constructor to create an accelerator table with the same entries as the specified one.

W32s Creating an instance of IAcceleratorTable with this constructor throws an exception.

© IBM Corp. 1992, 1996

IAcceleratorTable

Exceptions	
IInvalidRequest	Constructing an object of this class is not supported on the Win32s platform.

operator = Use this assignment operator to initialize an accelerator table with the same entries as the specified one.

```
IAcceleratorTable&
  operator =( const IAcceleratorTable& accelTbl );
```

Win	PM	Motif
Y	Y	N

accelTbl Existing IAcceleratorKey object whose keys are being copied.

~IAcceleratorTable

```
~IAcceleratorTable();
```

Win	PM	Motif
Y	Y	N

Handle Management

IAcceleratorTable allow you to convert an accelerator table object to an IAccelTblHandle (p. 42) object for calling APIs of the presentation system. IAcceleratorTable also gives you a way to manage the lifetime of the underlying system handle.

destroyHandle

Destroys the specified accelerator table handle.

The IAccelTblHandle class represents a system accelerator table handle, which is a system resource. Use this function to destroy the accelerator table handle created and returned by the function IAcceleratorTable::handle (p. 37). Only destroy an accelerator table handle when a window no longer needs the accelerator table.

Note: If you pass the accelerator table handle to IWindow::setAcceleratorHandle (p. 1048), you do not have to destroy it, since IWindow will manage it.

```
static void
  destroyHandle( IAccelTblHandle& accelTblHandle );
```

Win	PM	Motif
Y	Y	N

accelTblHandle
Handle of the accelerator table to destroy.

handle Converts the accelerator table into an accelerator table handle. If you call this function, you are responsible for destroying the accelerator table handle when it is no longer needed. See destroyHandle (p. 36) for details.

Generally you do not need to call this function. You can assign an accelerator table to a window by calling IWindow::setAcceleratorTable (p. 1048), which will manage destroying the handle for you.

```
IAccelTblHandle
  handle() const;
```

Win	PM	Motif
Y	Y	N

Querying Keys

IAcceleratorTable allows you to examine the collection of accelerator keys, including individual accelerator keys.

containsKeyLike

Identifies if the accelerator table contains an entry with the same keystroke (the same character or virtual key and the same Alt, Ctrl, Shift modifiers) as the specified accelerator key. If the table contains a matching entry, this function returns true. Otherwise it returns false.

```
Boolean
  containsKeyLike( const IAcceleratorKey& key ) const;
```

Win	PM	Motif
Y	Y	N

keyAt Returns a copy of the accelerator key at the cursor position. The cursor must be valid.

```
static IAcceleratorKey
  keyAt( const Cursor& cursor );
```

Win	PM	Motif
Y	Y	N

keyCount Returns the number of accelerator keys contained in the accelerator table.

```
unsigned long
  keyCount() const;
```

Win	PM	Motif
Y	Y	N

locateKeyLike

Searches the accelerator table for an entry with the same keystroke (the same character or virtual key and the same Alt, Ctrl, Shift modifiers) as the specified accelerator key. If the table contains a matching entry, this function returns true and points the cursor to the entry. Otherwise this function returns false.

© IBM Corp. 1992, 1996

IAcceleratorTable

```
Boolean
  locateKeyLike( const IAcceleratorKey& key,
                 Cursor& cursor ) const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IAcceleratorTable contains the following nested classes:

IAcceleratorTable::Cursor (see page 39)

IAcceleratorTable::Cursor

Derivation

IBase
  IVBase
    IAcceleratorTable::Cursor

Inherited By None.

Header File iacceltb.hpp

Members

Member	Page	Member	Page
Constructor	39	isValid	40
Cursor	39	setToFirst	40
element	40	setToNext	40
invalidate	40	~Cursor	39

The nested class IAcceleratorTable::Cursor iterates the accelerator keys contained in an accelerator table. You use this class to return the IAcceleratorKey (p. 22) objects contained in an IAcceleratorTable (p. 30).

Public Functions

Constructors

You can construct and destruct objects of the IAcceleratorTable::Cursor class. You cannot copy or assign IAcceleratorTable::Cursor objects because both the copy constructor and the assignment operator are private functions.

Cursor Constructs objects of the IAcceleratorTable::Cursor class. You can use the resulting object to enumerate or manipulate the accelerator keys contained in the specified accelerator table object.

```
Cursor( IAcceleratorTable& accelTbl );
```

Win	PM	Motif
Y	*Y*	*N*

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

Position

You use a cursor to point to an accelerator key in an accelerator table. To iterate the keys in a table, you can query the accelerator key pointed to by a cursor, and change the entry in the table that a cursor is pointing to.

element Returns a copy of the accelerator key object pointed at by the cursor. To use this function, the cursor must be in a valid state.

```
IAcceleratorKey
  element() const;
```

Win	PM	Motif
Y	Y	N

setToFirst Points the cursor to the first accelerator key in the accelerator table. If the table contains no accelerator keys, this function invalidates the cursor and returns false. Otherwise this function puts the cursor in a valid state and returns true.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	N

setToNext Points the cursor to the next accelerator key in the accelerator table. If the cursor was already pointing to the last accelerator key, this function invalidates the cursor and returns false. Otherwise this function returns true.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	N

Validation

A cursor must be in a valid state to be used in manipulating the accelerator keys in an accelerator table. While you can use the positioning members of this class to validate a cursor, you can also query whether a cursor is valid as well as invalidate it.

invalidate Invalidates the cursor. You must put the cursor back into a valid state using setToFirst (p. 40) before you can use it to access an accelerator key.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	N

isValid Indicates if the cursor is in a valid state. This function returns true if the cursor is valid. Otherwise it returns false.

Note: This function may return true if the cursor is in an undefined state.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	N

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IAccelTblHandle

Derivation

IBase
IHandle
IAccelTblHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	42
operator Value	43

The IAccelTblHandle class accesses accelerator tables. Accelerator tables are also referred to as shortcut tables. The class IAccelerator (p. 17) describes accelerator tables.

PM IAccelTblHandle is an alias for the Presentation Manager Toolkit type HACCEL.

Motif AIX does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IAccelTblHandle

Constructs objects of this class from an accelerator table handle (a value of type IHandle::Value), which defaults to 0.

```
IAccelTblHandle( Value haccel = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

© Copyright IBM Corp. 1992, 1996

operator Value

Returns the IHandle value.

```
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**	

© IBM Corp. 1992, 1996

IAnchorBlockHandle

Derivation

IBase
IHandle
IAnchorBlockHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	44
operator Value	45

The IAnchorBlockHandle class lets the system access an application's thread-specific data.

PM IAnchorBlockHandle is an alias for the OS/2 Programmer's Toolkit type HAB.

Motif AIX does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IAnchorBlockHandle

Constructs objects from an anchor block handle (a value of type IHandle::Value), which defaults to 0.

```
IAnchorBlockHandle( Value hab = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

© Copyright IBM Corp. 1992, 1996

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

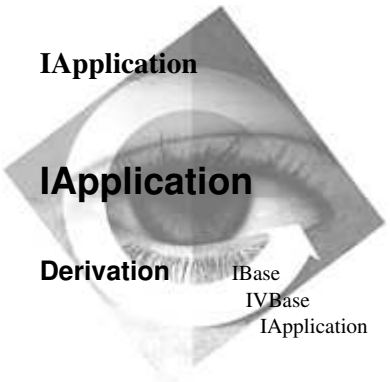
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IApplication

Derivation

IBase
IVBase
IApplication

Inherited By ICurrentApplication

Header File iapp.hpp

Members

Member	Page	Member	Page
Constructor	49	currentPID	49
adjustPriority	47	id	49
asDebugInfo	46	setId	50
asString	47	setPriority	48
current	48	~IApplication	50

The IApplication class represents processes. The User Interface Class Library only supports the currently executing application, the single object of the derived class ICurrentApplication (p. 237). IApplication::current provides access to that object.

This class maintains a static pointer to the C++ object representing the currently executing application. It does so to implement the static member function current (p. 48), which returns a reference to the ICurrentApplication object. The set of functions that you can apply to the current application is different from the set of functions you can apply to other applications. ICurrentApplication defines these functions.

Public Functions

Diagnostics

Use these members for diagnostic purposes. They return an IString representation of an object of this class.

asDebugInfo Returns a representation of the application as debug information. The representation is returned as an IString with the following contents:

```
IApplication(IVBase(@addr),id=pid)
```

This represents the following:

addr The address of the object (in hexadecimal).

© Copyright IBM Corp. 1992, 1996

pid The process ID (in hexadecimal).

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString

Returns the string "IApplication(pid)", where *pid* represents the process identifier.

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Priority

Use these members to control the priority of the application (and its threads).

Note: While you can set priorities using this class, only threads actually have a priority. As a result, you need to use functions of the IThread (p. 981) class to query the priority of an application's threads.

adjustPriority

Adjusts the priority level of all of the application's threads by some amount. An optional flag specifies whether the library also modifies the priority of descendent processes.

```
virtual IApplication&
  adjustPriority( long adjustment,
                  Boolean setDescendents = false );
```

Win	PM	Motif
I	Y	Y

adjustment Long value that represents the adjustment delta. This value must be in the range of -31 to 31.

setDescendents
Boolean that determines whether the library also modifies the priority of descendent processes.

W32s Win32s does not support multiple threads or thread priorities. Thus, IThread::adjustPriority and IApplication::adjustPriority have no effect.

Motif This member function calls IThread::adjustPriority for the current (and only) thread. The AIX release of the User Interface Class Library does not support multiple threads or thread priorities; thus, IThread::adjustPriority and IApplication::adjustPriority have no effect.

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The priority was not adjusted. The adjustment delta must be in the range of -31 to 31.

setPriority Sets the priority (p. 50) class and level of all of the application's threads to the specified value. An optional flag specifies whether the library also modifies the priority of descendent processes.

Win	PM	Motif
Y	Y	Y

```
virtual IApplication&
  setPriority( PriorityClass priorityClass,
               long priorityLevel = 0,
               Boolean setDescendents = false );
```

priorityClass
Enumeration that identifies the priority class to set.

priorityLevel
Long value that represents the new priority level of all the application's threads for the priority class. This value must be in the range of -31 to 31.

setDescendents
Boolean that determines whether the library also modifies the priority of descendent processes.

W32s Win32s does not support multiple threads or thread priorities. Thus, IThread::setPriority and IApplication::setPriority have no effect.

Motif This member function calls IThread::setPriority for the current (and only) thread. The AIX release of the User Interface Class Library does not support multiple threads or thread priorities; thus, IThread::setPriority and IApplication::setPriority have no effect.

Exceptions	
IAccessError	The priority was not set. The priority level may be invalid.

Process Information

Use these members to get additional information about a process, such as the process identifier, or to access the object for the current process.

current Returns a reference to the current application, which is an object of the class ICurrentApplication (p. 237).

```
static ICurrentApplication&
  current();
```

Win	PM	Motif
Y	Y	Y

currentPID Returns the current process identifier value.

```
static IProcessId
  currentPID();
```

Win	PM	Motif
Y	Y	Y

id Returns this object's process identifier value.

```
virtual IProcessId
  id() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

The constructors and destructor for this class are protected. You must derive from this class if you want to use it to represent an application.

IApplication You can only construct objects of this class with the process identifier for the process that the object will represent.

```
IApplication( const IProcessId& id );
```

Win	PM	Motif
Y	Y	Y

id Reference to the process identifier for the process that the object will represent.

© IBM Corp. 1992, 1996

IApplication

~IApplication

```
virtual
  ~IApplication();
```

Win	PM	Motif
Y	Y	Y

Setting Process Information

Use these members to set the information on a process, such as the process identifier of this object.

setId Sets this object's process identifier. You call this function in your derived class when you start a process in order to save the process identifier for this object.

```
virtual IApplication&
  setId( const IProcessId& id );
```

Win	PM	Motif
Y	Y	Y

id Reference to the process identifier to save.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

PriorityClass

```
PriorityClass {
    noChange,        idleTime,        regular,         timeCritical,
    foregroundServer
    };
```

These enumerators specify one of the priority classes, such as when using setPriority: (p. 48)

noChange
Specifies no change to the thread priority.

idleTime
Specifies the lowest priority. This priority only gets processing time when there is no other work to do.

regular
Specifies the default priority. Most threads belong to this class.

timeCritical
Specifies the highest priority. Use this priority when response time is critical.

foregroundServer

Specifies that a program running on a server takes precedence over other processes on the server.

PM For additional information, see the *OS/2 3.0 Control Program Guide and Reference*.

W32s The library provides this enumeration for portability purposes only. Win32s does not support priority adjustment.

Motif The library provides this enumeration for portability purposes only. The AIX release of the User Interface Class Library does not support priority adjustment.

© IBM Corp. 1992, 1996

IBaseComboBox

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IEntryField
IBaseComboBox

Inherited By

ICollectionViewComboBox
IComboBox

Header File

icombobs.hpp

Members

Member	Page	Member	Page
Constructor	55	minimumRows	59
anyData	70	mixedData	71
autoScroll	70	moveSizeTo	65
border3D	70	nativeRect	65
calcMinimumSize	68	notFound	69
changeCount	67	numberOfSelections	62
classDefaultStyle	70	oemData	71
convertToGUIStyle	63	position	65
count	56	readOnlyDropDownType	71
dbcsData	70	registerCallbacks	67
defaultStyle	63	sbcsData	71
deselect	61	select	62
deselectAll	62	selectId	69
dropDownType	70	selection	63
elementAt	57	setBackgroundColor	54
enableNotification	60	setDefaultStyle	63
enterId	69	setForegroundColor	55
first	69	setItemHandle	58
hasFocus	54	setItemText	57
hideList	60	setLayoutDistorted	68
horizontalScroll	71	setLimit	64
incrementChangeCount	67	setMinimumRows	59
isEmpty	57	setTop	60
isHorizontalScroll	63	showList	60
isListShowing	60	simpleType	72
isSelected	62	size	65
itemHandle	58	top	60
itemText	57	topHandle	55
layoutAdjustment	59	type	64
limit	64	unregisterCallbacks	67
locateText	61	visibleRectangle	59

© Copyright IBM Corp. 1992, 1996

IBaseComboBox

Member	Page
~IBaseComboBox	56

The IBaseComboBox class creates and manages combination box control windows. The IBaseComboBox class combines an entry field and a list box to form one control containing the features of both. IBaseComboBox presents a general combination box interface except for population capability, such as add, remove, or replace items. The derived classes IComboBox (p. 182) and ICollectionViewComboBox (Vol. III) supply these functions. IComboBox contains the add, remove, replace functions, while ICollectionViewComboBox populates a combination box's list box from collection elements via setItems. Typically, you use one of these derived classes of IBaseComboBox.

Handlers derived from the following classes handle events for IBaseComboBox objects:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IResizeHandler (p. 802)
- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- ISelectHandler (p. 851)
- IShowListHandler (p. 870)

PM The parent of an IBaseComboBox should not set the IWindow::clipChildren style. This style prevents the OS/2 operating system from painting a small region below the list box portion of the IBaseComboBox object.

Motif The IBaseComboBox constructor creates objects of this class using several Motif widgets. An XmForm widget is created with XmText, XmScrolledWindow, and XmList children. If the IBaseComboBox object has the dropDownType or the readOnlyDropDownType styles, an XmArrowButton is also created as a child of the XmForm widget. IWindow::handle (p. 1049) returns the handle of the XmText widget.

Note: The IBaseComboBox member functions use 0-based indexes, rather than the 1-based indexes used by the Motif XmList functions.

The behavior of an IBaseComboBox object is provided by private callbacks and a default handler. The IBaseComboBox class uses a default handler attached to the IBaseComboBox object. Therefore, attach user-defined handlers to the IBaseComboBox rather than to its owner window. Doing so enables events to be dispatched to user-defined handlers before the default handler.

© IBM Corp. 1992, 1996

Handlers derived from IEditVerifyHandler can be attached to IBaseComboBox objects.

The IBaseComboBox class does not exist in the User Interface Class Library product on Motif platforms. In those versions of the User Interface Class Library, the IComboBox class provides the entire interface. This version of User Interface Class Library splits the interface into the IComboBox and IBaseComboBox classes.

AIX only supports the constructor that creates an object of this class using the control ID, parent window, owner window, rectangle, and style parameters.

To process keystrokes for the OS/2 platform, you must attach an IKeyboardHandler (p. 490) or IEditHandler (p. 267) to the entry field child of the combination box object. You must wrapper the entry field using one of the IEntryField constructors.

For the AIX platform, you can attach an IKeyboardHandler or IEditHandler directly to the IBaseComboBox object. Do not wrapper the entry field within the combination box. You might want to use the IEditVerifyHandler instead of IKeyboardHandler for processing entry field changes.

Public Functions

Attributes

Use these members to query and change characteristics of the entry field control.

hasFocus

If the window has the input focus, true is returned. Otherwise, false is returned.

```
virtual Boolean
  hasFocus() const;
```

Win	PM	Motif
Y	*N*	*N*

Colors

Use these members to access the colors of the IBaseComboBox.

setBackgroundColor

Sets the background color to the specified color.

```
virtual IBaseComboBox&
  setBackgroundColor( const IColor& color );
```

Win	PM	Motif
N	*N*	*Y*

setForegroundColor

Sets the foreground color to the specified color.

```
virtual IBaseComboBox&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
N	N	Y

Compound Control

Use these members to access the top window system.

topHandle

Returns the topmost manager widget of the combination box.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	N	Y

Constructors

You can construct and destruct objects of this class.

IBaseComboBox

1

```
IBaseComboBox( unsigned long id,
               IWindow* parent );
```

Win	PM	Motif
Y	Y	N

You can construct objects of this class using the parent window and a combination box ID.

id A combination box control ID.
parent The parent window.

Motif In the AIX environment, the IBaseComboBox constructor creates objects of this class using several Motif widgets. An XmForm widget is created with XmText, XmScrolledWindow, and XmList children. If the IBaseComboBox object has the dropDownType or the readOnlyDropDownType styles, an XmArrowButton is also created as a child of the XmForm widget. Use IWindow::handle to return the handle of the XmText widget.

The behavior of an IBaseComboBox object is accomplished via private callbacks and a default handler. The default handler used by the IBaseComboBox class is attached to the IBaseComboBox object; thus, attach user-defined handlers to the IBaseComboBox object rather than to its owner. This enables events to be dispatched to user-defined handlers before the default handler.

© IBM Corp. 1992, 1996

IBaseComboBox

2 `IBaseComboBox( unsigned long id,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial = IRectangle ( ),`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class using the ID, parent, owner, size, position, and style parameters.

id A combination box control ID.

parent The parent window.

owner The owner window.

initial The initial position and size of the combination box you construct. The default is the rectangle constructed by the default IRectangle constructor. Optional.

style The combination box's characteristics. Optional.

Exceptions	
IInvalidParameter	Either the parent was NULL or an invalid style was passed.

3 `IBaseComboBox( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	N

You can construct objects of this class using the handle of an existing combination box window.

handle The window handle of an existing combination box control.

Motif The Motif implementation of IBaseComboBox does not use a singular Motif widget; therefore, you cannot construct objects of this class from the handle of a combination box control in the AIX environment.

~IBaseComboBox

`virtual`
`~IBaseComboBox();`

Win	PM	Motif
Y	Y	Y

Content

Use these members to determine the number of items in the list box portion of the combination box.

count Returns the number of items in the list box.

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

Win	PM	Motif
Y	Y	Y

isEmpty If the list box is empty, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isEmpty() const;
```

Win	PM	Motif
Y	Y	Y

Get and Change Items

Use these members to retrieve or change a text item in the list box portion of the combination box.

elementAt Returns the string of the item at the cursor position.

```
virtual IString
  elementAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The cursor is invalid

itemText Returns the text of the specified item in the list box portion of the combination box.

index Index of the item in a list box.

```
virtual IString
  itemText( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The index is invalid.

setItemText Changes the text of the specified item in the list box portion of the combination box.

index Index of the item in a list box.
string The new text.
updateEntryField
If the specified item in the list box is selected and this flag is true, the combination box entry field gets updated.

© IBM Corp. 1992, 1996

IBaseComboBox

1
```
virtual IBaseComboBox&
  setItemText( unsigned long index,
               const IResourceId& string,
               Boolean updateEntryField = false );
```

Win	PM	Motif
Y	Y	Y

Win The updateEntryField parameter is ignored in the Windows environment. The entry field portion of the combo box will always be updated with the new item text if the selected item is changed.

2
```
virtual IBaseComboBox&
  setItemText( unsigned long index,
               const char* string,
               Boolean updateEntryField = false );
```

Win	PM	Motif
Y	Y	Y

Win The updateEntryField parameter is ignored in the Windows environment. The entry field portion of the combo box will always be updated with the new item text if the selected item is changed.

Exceptions	
IAccessError	The operating system is unable to set the text of the list box item.

Handle Members

Use these members to set or retrieve a handle of a list box item.

itemHandle Returns the handle of the specified list box item. If the item does not have a handle, 0 is returned.

index Index of the item in a list box.

```
virtual unsigned long
  itemHandle( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

setItemHandle

Sets the handle of the specified list box item.

index Index of the item in a list box.
handle Handle of the list box item.

```
virtual IBaseComboBox&
  setItemHandle( unsigned long index,
                 unsigned long handle );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system is unable to set the handle of the list box item.

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

layoutAdjustment

Returns the dimensions that a window should be moved or sized to after a canvas runs its layout routines. If the combination box has a drop-down list box, this function returns an IRectangle that adjusts the combination box by the size of the drop-down list box.

```
virtual IRectangle
  layoutAdjustment() const;
```

Win	PM	Motif
Y	Y	Y

minimumRows

Returns the number of visible rows in the list box portion of a minimum size combination box window.

```
unsigned long
  minimumRows() const;
```

Win	PM	Motif
Y	Y	Y

setMinimumRows

Sets the number of rows in the list box portion of a minimum size combination box window. If you do not call this function, four rows are shown.

minimumRows
Minimum number of visible rows.

```
virtual IBaseComboBox&
  setMinimumRows( unsigned long minimumRows );
```

Win	PM	Motif
Y	Y	Y

visibleRectangle

Returns the window rectangle that is painted by the control.

```
virtual IRectangle
  visibleRectangle() const;
```

Win	PM	Motif
Y	Y	Y

List Box Operations

Use these members to query, show, or hide the list box portion of the combination box.

© IBM Corp. 1992, 1996

hideList Hides the list box.

```
virtual IBaseComboBox&
  hideList();
```

Win	PM	Motif
Y	Y	Y

isListShowing

If the list box is visible, true is returned. Otherwise, false is returned.

```
Boolean
  isListShowing() const;
```

Win	PM	Motif
Y	Y	Y

showList Shows or hides the list box. This is only valid for combination boxes with the dropDownType or readOnlyDropDownType styles.

```
virtual IBaseComboBox&
  showList( Boolean show = true );
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables or disables the combination box to send notifications to any observer objects.

```
virtual IBaseComboBox&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Scrolling Members

Use these members to scroll an item to the top of the list box or to return the item at the top of the list box.

setTop Scrolls the specified item to the top of the list box.

index Index of the item in a list box.

```
virtual IBaseComboBox&
  setTop( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

Win This member function has no effect on Windows NT 3.51 with Program Manager.

top Returns the item number of the item currently at the top of the list box.

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

Win	PM	Motif
Y	Y	Y

Win This member function always returns 0 on Windows NT 3.51 with Program Manager.

Exceptions	
IAccessError	There are no items in the list box.

Search List

Use these members to search the list box portion of the combination box for a text string. The data structures aid that effort.

locateText Returns the item number of the list box item matching the search string. The search starts after the specified index. If no match is found, notFound is returned.

searchString
String to search for.

caseSensitive
Indicator of whether the search is case-sensitive. The default of true means the search is case-sensitive.

search Use the enumeration SearchType (p. 73) to specify the type of search to perform. The default is exactMatch.

index Index of where to start the search. The default is first.

```
virtual unsigned long
  locateText( const char* searchString,
              Boolean caseSensitive = true,
              SearchType search = exactMatch,
              unsigned long index = first ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system is unable to search the list box for the text.

Selection

Use these members to set or query the list box's selection state.

deselect Removes the selection state from the specified item.

index 0-indexed reference to item.

© IBM Corp. 1992, 1996

```
virtual IBaseComboBox&
  deselect( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

deselectAll Removes the selection state from the currently selected item in the list box.

```
virtual IBaseComboBox&
  deselectAll();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system is unable to deselect all of the items in the list box.

isSelected If the specified item is currently selected, true is returned. Otherwise, false is returned.

index Index of the item in a list box.

```
virtual Boolean
  isSelected( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

numberOfSelections

If no item is selected, 0 is returned. Otherwise, 1 is returned. The current selection does not change.

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

Win	PM	Motif
Y	Y	Y

select Sets the selection state of the specified item.

index The index of the text item.
select If you specify true, the item is selected and any previously selected item is deselected. If you specify false, the item is deselected. The default is true.

```
virtual IBaseComboBox&
  select( unsigned long index,
          Boolean select = true );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system is unable to select the list box item.

selection Returns the 0-based index of the selected item. If no item is selected, notFound is returned.

```
virtual long
  selection() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to set and query combo box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IEntryField Styles (p. 290)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 70) unless you have changed it using setDefaultStyle (p. 63).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

isHorizontalScroll

If the list box part of the combination box has a horizontal scroll bar, true is returned. Otherwise, false is returned.

```
Boolean
  isHorizontalScroll() const;
```

Win	PM	Motif
Y	Y	Y

Win The horizontalScroll style is ignored on Windows NT 3.51 with Program Manager. Thus, this member function always returns false in that environment.

setDefaultStyle

Sets the default style for all subsequent combination boxes.

© IBM Corp. 1992, 1996

style Use the styles provided by IBaseComboBox Styles (p. 69) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Text Limit

Use these members to set and query the text limit of the entry field.

limit Returns the length, in bytes, of the longest text the entry field can hold.

Note: The default value for the limit differs from system to system in accordance with the look-and-feel of that system. If your application requires a specific limit for an entry field, you must set that limit using setLimit to ensure portability of your application.

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

Win	PM	Motif
Y	N	N

setLimit Sets the maximum number of bytes the entry field can contain.

The calcMinimumSize function uses the text limit to determine the minimum size of the entry field.

Note: The default value for the limit differs from system to system in accordance with the look-and-feel of that system. If your application requires a specific limit for an entry field, you must set that limit using setLimit to ensure portability of your application.

1
```
virtual IBaseComboBox&
  setLimit( unsigned long textLimit );
```

Win	PM	Motif
Y	N	N

2
```
virtual IBaseComboBox&
  setLimit( const IResourceId& textLimit );
```

Win	PM	Motif
Y	N	N

Type Members

Use these members to query the type of the combination box. The types correspond to similarly named styles.

type Returns the ControlType (p. 73) enumerator for the type of combination box.

```
ControlType
  type() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Window Positioning

Use these members to set and query the size and position of windows.

moveSizeTo Changes the position and size of the combination box window using a coordinate system that is lower-left origin-based.

aRectangle A rectangle for the combination box window.

```
virtual IBaseComboBox&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

nativeRect Returns a rectangle representing the position and size of the window. Unlike IWindow::rect (p. 1079), this function always returns the position in the native GUI orientation.

```
virtual IRectangle
  nativeRect() const;
```

Win	PM	Motif
Y	*N*	*N*

This member is overridden in this derived class for specific operating system behavior.

position Returns the combination box window's position using a coordinate system that is lower-left origin-based.

```
virtual IPoint
  position() const;
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

size Returns the current size of the combination box.

```
virtual ISize
  size() const;
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

© IBM Corp. 1992, 1996

Inherited Public Functions

IEntryField		
alignment	enable	isWriteable
backgroundColor	enableAutoScroll	leftIndex
charType	enableAutoTab	limit
clear	enableCommand	moveSizeTo
convertToGUIStyle	enableDataUpdate	paste
copy	enableInsertMode	removeAll
cursorPosition	enableMargin	resetTextChangedFlag
cut	enableNotification	selectedRange
defaultStyle	foregroundColor	selectedText
disable	hasSelectedText	selectedTextLength
disableAutoScroll	hasTextChanged	selectRange
disableAutoTab	isAutoScroll	setAlignment
disableCommand	isAutoTab	setCharType
disableDataUpdate	isCommand	setCursorPosition
disableInsertMode	isEmpty	**setDefaultStyle**
disableMargin	isInsertMode	setLeftIndex
discard	isMargin	setLimit

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Registers all possible callbacks and X event handlers to this object for events it might receive. IHandler derived classes later determine which events they will process.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Unregisters Motif callbacks for the widgets created during the construction of objects of this class.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Item Changes

Use these members to set or retrieve the number of changes (add or remove items) that have occurred to the list box portion of the combination box. These members help track the validity of IBaseComboBox::Cursor objects.

changeCount Retrieves the number of changes to the combination box.

```
unsigned long
  changeCount() const;
```

Win	PM	Motif
Y	*Y*	*Y*

incrementChangeCount

Increments the count of the number of changes to the list box portion of the combination box. IBaseComboBox-derived classes doing add and remove operations

© IBM Corp. 1992, 1996

should call this function because add and remove functions may cause an IBaseComboBox::Cursor to become invalid.

```
void
  incrementChangeCount();
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size of the combination box.

The size dimensions are as follows:

width The minimum entry field width.

height The maximum character height multiplied by the minimum number of rows. The minimum number of rows is set by calling setMinimumRows. If you do not call setMinimumRows, a default of four rows is used.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

setLayoutDistorted

Indicates that changes have occurred in the window causing the layout of the window in a canvas to be updated.

```
virtual IBaseComboBox&
  setLayoutDistorted( unsigned long layoutAttributeOn,
                      unsigned long layoutAttributeOff );
```

Win	PM	Motif
N	N	Y

Inherited Protected Functions

IEntryField		
calcMinimumSize	isDragStarting	registerCallbacks
initialize	passEventToOwner	setLayoutDistorted

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enterId Notification identifier provided to observers when an item in the list box portion of a combination box is double-clicked on or Enter is pressed for a selected item.

```
static INotificationId const
  enterId;
```

Win	PM	Motif
Y	*Y*	*Y*

selectId Notification identifier provided to observers when an item is selected in the list box portion of a combination box.

```
static INotificationId const
  selectId;
```

Win	PM	Motif
Y	*Y*	*Y*

Search List

Use these members to search the list box portion of the combination box for a text string. The data structures aid that effort.

first Searches from the beginning of the string. It is a value for the index parameter of IBaseComboBox::locateText.

```
static const unsigned long
  first;
```

Win	PM	Motif
Y	*Y*	*Y*

notFound Indicates no match is found. IBaseComboBox::locateText returns this value.

```
static const unsigned long
  notFound;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for this class. Use these members to set and query combo box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IEntryField Styles (p. 290)

© IBM Corp. 1992, 1996

IBaseComboBox

anyData Sets the combination box entry field to accept text that is a mixture of SBCS and DBCS characters.

Note: If the text contains both single-byte and double-byte characters and will be converted from an ASCII code page into an EBCDIC code page, this style causes an entry field to ignore accounting for shift-in and shift-out characters that would be introduced into its text. This style is the opposite of mixedData.

```
static const Style
  anyData;
```

Win	PM	Motif
N	*Y*	*I*

autoScroll If the user tries to move off the end of a line, the entry field automatically scrolls one-third the width of the window in the appropriate direction.

```
static const Style
  autoScroll;
```

Win	PM	Motif
Y	*Y*	*N*

border3D Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	*I*	*I*

Win This style is always present in the Windows NT and Windows 95 environments.

Win32s This style is ignored on Win32s.

classDefaultStyle

Provides the original default style for this class, which is the following: IBaseComboBox::simpleType | IBaseComboBox::anyData | IBaseComboBox::border3D | IBaseComboBox::autoScroll | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

dbcsData Sets the combination box's entry field to accept double-byte characters only.

```
static const Style
  dbcsData;
```

Win	PM	Motif
N	*Y*	*I*

dropDownType

Specifies the drop-down variation of the combination box, which is the same as simpleType except that the list box control is hidden until the user requests that it be displayed.

```
static const Style
  dropDownType;
```

Win	PM	Motif
Y	*Y*	*Y*

horizontalScroll

Provides horizontal scrolling for the list box control.

```
static const Style
  horizontalScroll;
```

Win	PM	Motif
Y	*Y*	*Y*

Win This style is ignored on Windows NT 3.51 with Program Manager.

mixedData

Sets the combination box's entry field to accept text that is a mixture of SBCS and DBCS characters. Conversion from an ASCII DBCS code page to an EBCDIC DBCS code page can result in a possible increase in the length of the data because of the addition of shift in/shift out (SI/SO) characters, but it will not exceed the text limit of the entry field.

Note: This style is the opposite of anyData.

```
static const Style
  mixedData;
```

Win	PM	Motif
N	*Y*	*I*

oemData

Allows Windows to properly convert the text into a specific character set when a new code page is used.

```
static const Style
  oemData;
```

Win	PM	Motif
Y	*N*	*I*

readOnlyDropDownType

Specifies the drop-down list variation of the combination box. The drop-down list does not allow a user to type information into the entry field. The drop-down list only displays one item in the entry field until the user causes the display of the list box control to make alternative selections in the list box. The entry field control is replaced by a static control that displays the current selection from the list box.

```
static const Style
  readOnlyDropDownType;
```

Win	PM	Motif
Y	*Y*	*Y*

sbcsData

Sets the combination box's entry field to accept single-byte characters only.

```
static const Style
  sbcsData;
```

Win	PM	Motif
N	*Y*	*I*

© IBM Corp. 1992, 1996

simpleType Specifies that both the entry field control and the list box control are visible. When the selection changes in the list box control, the text of the selected item in the list box control is placed in the entry field. Also, the user can select an item from the list box control by typing a portion of the item into the entry field. The entry field is filled with the closest match from the list box.

```
static const Style
  simpleType;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IEntryField		
characterTypeId	dataUpdateId	end

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IBaseComboBox contains the following nested classes:

IBaseComboBox::Style (see page 79)
IBaseComboBox::Cursor (see page 75)

ControlType

```
ControlType {
    simple,
    dropDown,
    dropDownList,
    readOnlyDropDown = dropDownList
    };
```

Use these enumerators to specify the types of combination boxes:

simple
Specifies that both the entry field control and the list box control are visible.

When the user selects an item in the list box, the IComboBox object places the text of the selected item in the entry field. The user can also select an item in the list box by typing a portion of the item into the entry field. IComboBox completes the entry with the closest matching item in the list box.

dropDown
Specifies the drop-down variation of the combination box. In addition to all of the properties of a simple combination box control, the drop-down list box is hidden until the user requests that it be displayed.

readOnlyDropDown
Specifies the read-only drop-down variation of the combination box. The read-only drop-down does not let a user type information into the entry field.

SearchType

```
SearchType {
    prefix,
    substring,
    exactMatch
    };
```

Use these enumerators to specify the type of search:

© IBM Corp. 1992, 1996

prefix
If the leading characters of the item contain the specified characters, a match occurs.

substring
If the item contains a substring of the specified characters, a match occurs.

exactMatch
If the item is an exact match for the specified characters, a match occurs.

IBaseComboBox::Cursor

Derivation

IBase
  IVBase
    IBaseComboBox::Cursor

Inherited By None.

Header File icombobs.hpp

Members

Member	Page	Member	Page
Constructor	75	setToIndex	76
asIndex	77	setToLast	76
Cursor	75	setToNext	76
invalidate	77	setToPrevious	76
isValid	77	~Cursor	76
setToFirst	76		

The nested class IBaseComboBox::Cursor defines objects that you can use to traverse through the items in a combination box. In the same way that you can use a cursor to traverse through the objects in a collection, you can use this cursor to traverse through a combination box, one item at a time.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor You can create objects of this class by using the combination box and a filter type.

comboBox A combination box control.

type Use the enumeration Filter (p. 77) to specify the type of filter for the combination box. The type specifies how the cursor traverses to the next item in the list box portion of the combination box. The default is selectedItems.

```
Cursor( const IBaseComboBox& comboBox,
        Filter type = selectedItems );
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

IBaseComboBox::Cursor

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Cursor Movement

Use these members to control cursor movement in the list box portion of a combination box.

setToFirst Points to the first list box item and validates the cursor. If the cursor is set successfully, true is returned. Otherwise, false is returned.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToIndex Points to the specified item and validates the cursor. If the cursor is set to the specified list box item, true is returned. Otherwise, false is returned.

index A 0-based index indicating the item upon which you want the cursor placed.

```
virtual Boolean
  setToIndex( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

setToLast Points to the last list box item and validates the cursor. If the cursor is set successfully, true is returned. Otherwise, false is returned.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Points to the next item in the list box. If the cursor is set to the next list box item, true is returned. If no more items exist, this function invalidates the cursor and false is returned.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Points to the previous item in the list box. If the cursor is set to the previous list box item, true is returned. If no previous item exists, this function invalidates the cursor and false is returned.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation and Conversion

Use these members to query or return information about the item to which the cursor is pointing.

asIndex Returns the 0-based index of the item pointed to by the cursor.

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

Win	PM	Motif
Y	Y	Y

invalidate Flags the cursor as not valid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Returns a Boolean depending on whether the cursor is pointing to a valid item. If the cursor is pointing to a valid item, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IBaseComboBox::Cursor

```
Filter {
    selectedItems,
    allItems
    };
```

Use these enumerators to specify the filter for the combination box cursor:

selectedItems
Filters only items that are selected.

allItems
Filters all items.

IBaseComboBox::Style

Derivation

IBase
  IBitFlag
    IBaseComboBox::Style

Inherited By None.

Header File icombobs.hpp

The nested class IBaseComboBox::Style provides a set of valid styles for the IBaseComboBox (p. 52) class.

PM If you need to change the style of the entry field region of the combination box and the style is not available through the previously listed style classes, you can create an IEntryField object for the entry field child of the combination box and then use the members provided by IEntryField (p. 290) Styles.

The AIX version ignores the dbcsData, mixedData, and sbcsData styles; only the anyData style is supported for text type.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

© Copyright IBM Corp. 1992, 1996

IBaseComboBox::Style

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IBaseListBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IBaseListBox

Inherited By

ICollectionViewListBox
IListBox

Header File

ilistbas.hpp

Members

Member	Page	Member	Page
Constructor	84	isSelected	90
backgroundColor	83	itemHandle	86
border3D	98	itemHeight	87
calcMinimumSize	96	itemText	85
changeCount	96	locateText	89
classDefaultStyle	98	minimumCharacters	87
convertToGUIStyle	91	minimumRows	87
count	85	multipleSelect	99
defaultStyle	91	noAdjustPosition	99
deselect	89	notFound	97
deselectAll	90	numberOfSelections	90
disableDrawItem	92	passEventToOwner	95
disableExtendedSelect	92	registerCallbacks	95
disableMultipleSelect	92	select	90
disableNoAdjustPosition	92	selectAll	91
drawItem	98	selectId	97
elementAt	85	selection	91
enableDrawItem	92	setDefaultStyle	94
enableExtendedSelect	92	setItemHandle	86
enableMultipleSelect	93	setItemHeight	87
enableNoAdjustPosition	93	setItemText	86
enableNotification	88	setLayoutDistorted	87
enterId	97	setMinimumCharacters	88
extendedSelect	98	setMinimumRows	88
first	97	setTop	88
horizontalScroll	99	show	94
incrementChangeCount	96	top	89
isDrawItem	93	topHandle	84
isEmpty	85	unregisterCallbacks	96
isExtendedSelect	93	~IBaseListBox	85
isHorizontalScroll	93		
isMultipleSelect	94		
isNoAdjustPosition	94		

© Copyright IBM Corp. 1992, 1996

IBaseListBox

The IBaseListBox class creates and manages list box control windows. List boxes always have vertical scroll bars. Optionally, they can have horizontal scroll bars. IBaseListBox contains general list box function except for population capability, such as adding, removing, or replacing list box items. The derived classes IListBox (p. 495) and ICollectionViewListBox (Vol. III) supply these functions. IListBox contains the add, remove, and replace functions, while ICollectionViewListBox populates a list box from collection elements via the setItems member function. Typically, you use one of these classes derived from IBaseListBox.

You can enable a list box for the following types of item selection techniques:

single selection
The user can select only one item at a time.

multiple selection
The user can select any number of items or not select any.

extended selection
The user can extend selection to more than one item.

These three techniques are mutually exclusive in one list box.

A list box operates as if it is a 0-based array of items. The item index requested or returned is the 0-based index number of the location of the item in question.

Handlers derived from the following classes handle events for IBaseListBox:

- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IListBoxDrawItemHandler (p. 513)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISelectHandler (p. 851)

PM When you add an IBaseListBox as a child window of a canvas and use ISetCanvas, ISplitCanvas, or IMultiCellCanvas, you must give it the style IBaseListBox::noAdjustPosition (p. 99).

Motif The User Interface Class Library implements the IBaseListBox class using the XmList widget. Motif application developers should note that the IBaseListBox member functions use 0-based indexes, unlike the Motif XmList functions, which use 1-based indexes.

The library creates scrolled list box controls using the XmCreateScrolledList convenience function. This convenience function creates a composite control consisting of an XmScrolledWindow parent with an XmList child. The IBaseListBox control's handle is the widget ID of the XmList child, and any application utilizing native Motif functions should reflect this.

Two selection styles, extendedSelect and multipleSelect, are available for IBaseListBox controls. There is no explicit single-selection mode style. In the AIX environment, the default selection style is the Motif XmList default selection policy, XmBROWSE_SELECT. This default selection policy is not forced by the User Interface Class Library. Therefore, you can use an application resource file to specify your desired XmNselectionPolicy. By using an application resource file, you can specify an XmNselectionPolicy of XmSINGLE_SELECT.

The IBaseListBox class does not exist in the User Interface Class Library product on Motif platforms. In those versions of the User Interface Class Library, the IListBox class provides the entire interface. This version of User Interface Class Library splits the interface into the IListBox and IBaseListBox classes.

Motif does not support owner-draw list boxes. This is because the underlying XmList widget treats the list box entries as an array of textual data. The OS/2 operating system Presentation Manager list box control, on the other hand, accesses each list box entry as a separate item.

Public Functions

Colors

Use these members to access the colors of the IBaseListBox.

backgroundColor

Returns the background color value of the list box area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	*Y*	*N*

This member is overridden in this derived class for specific operating system behavior.

Compound Control

Use these members to return miscellaneous information for this compound control.

© IBM Corp. 1992, 1996

topHandle

Returns the top manager widget of the list box. The IWindowHandle (p. 1107) returned is used for sizing, positioning, and managing the list box.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	N	Y

Constructors

You can construct and destruct objects of this class.

IBaseListBox

1

```
IBaseListBox( unsigned long id,
              IWindow* parent,
              IWindow* owner,
              const IRectangle& initial = IRectangle ( ),
              const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the parent window, owner window, optional size and location, and optional style arguments.

id The ID of a list box control.

parent The parent window.

owner The owner window.

initial A rectangle for the list box control. It specifies the initial position and size of the IBaseListBox you construct. The initial position is the lower-left corner of the list box relative to the lower-left corner of the parent window. The default is the rectangle constructed by the default IRectangle constructor. Optional.

style The initial style for the list box control. The classDefaultStyle is used if no style is provided. Optional.

Exceptions	
IInvalidParameter	The parent window is not valid.

2

```
IBaseListBox( unsigned long id,
              IWindow* parent );
```

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the parent window and a list box control ID.

id The ID of a list box control.

parent The parent window.

3 `IBaseListBox( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the handle of an existing list box control.

handle A window handle of an existing list box control.

~IBaseListBox

```
virtual
  ~IBaseListBox();
```

Win	PM	Motif
Y	Y	Y

Content

Use these members to determine the number of items in the list box.

count Returns the number of items in the list box.

Note: You can use IBaseListBox::count and isSelected to process all of the selected list box items. However, it is easier to use the cursor facility provided by IBaseListBox::Cursor (p. 101).

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

Win	PM	Motif
Y	Y	Y

isEmpty If the list box is empty, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isEmpty() const;
```

Win	PM	Motif
Y	Y	Y

Get and Change Items

Use these members to retrieve or change the text of an item in the list box.

elementAt Returns the string of the item at the cursor position.

```
virtual IString
  elementAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

itemText Returns the text of the specified item in the list box.

© IBM Corp. 1992, 1996

index Index of the item in a list box.

```
virtual IString
  itemText( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

setItemText Changes the text of the specified item in the list box.

index Index of the item in a list box.
string The new text.

1
```
virtual IBaseListBox&
  setItemText( unsigned long index,
               const IResourceId& string );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual IBaseListBox&
  setItemText( unsigned long index,
               const char* string );
```

Win	PM	Motif
Y	Y	Y

Handle Members

Use these members to set or retrieve a handle of a list box item. A *handle* is an optional data field that is associated with each list box item.

itemHandle Returns the handle of the specified list box item. If the item does not have a handle, 0 is returned.

index Index of the item in a list box.

```
virtual unsigned long
  itemHandle( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

setItemHandle

Sets the handle of the specified list box item.

index Index of the item in a list box.
handle Handle of the list box item.

```
virtual IBaseListBox&
  setItemHandle( unsigned long index,
                 unsigned long handle );
```

Win	PM	Motif
Y	Y	Y

Item Height

Use these members to set or retrieve a list box item's height.

itemHeight

Returns the height, in pixels, of a list box item. This is not the font size. It is the height of the cell within which an item is displayed.

```
unsigned long
  itemHeight() const;
```

Win	PM	Motif
Y	N	Y

Exceptions	
IAccessError	The operating system's request to query the item's height has failed.

setItemHeight

Sets the height, in pixels, of a list box item. This does not change the font size. It changes the height of the cell within which an item is displayed.

```
virtual IBaseListBox&
  setItemHeight( unsigned long newHeight );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to set the height has failed.

Layout Support

Use these members to support canvases.

minimumCharacters

Returns the maximum number of characters in an item of a minimum size list box.

```
unsigned long
  minimumCharacters() const;
```

Win	PM	Motif
Y	Y	Y

minimumRows

Returns the number of visible rows of a minimum size list box.

```
unsigned long
  minimumRows() const;
```

Win	PM	Motif
Y	Y	Y

setLayoutDistorted

Causes a parent ISetCanvas (Vol. III) or IMultiCellCanvas (Vol. III) to update the size and position of the list box when it is assigned a new font.

© IBM Corp. 1992, 1996

```
virtual IBaseListBox&
  setLayoutDistorted( unsigned long layoutAttributesOn,
                      unsigned long layoutAttributesOff );
```

Win	PM	Motif
Y	Y	N

setMinimumCharacters

Sets the maximum number of characters in an item of a minimum size list box. If you do not call this function, an item width of 25 characters is used.

```
virtual IBaseListBox&
  setMinimumCharacters( unsigned long minimumCharacters );
```

Win	PM	Motif
Y	Y	Y

setMinimumRows

Sets the number of rows in a minimum size list box. If you do not call this function, four rows are shown.

minimumRows
Minimum number of visible rows.

```
virtual IBaseListBox&
  setMinimumRows( unsigned long minimumRows );
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables the list box to send notifications to any observer objects.

```
virtual IBaseListBox&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Scrolling Members

Use these members to scroll an item to the top of the list box or to return the item at the top of the list box.

setTop

Scrolls the specified item to the top of the list box.

```
virtual IBaseListBox&
  setTop( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	There are no items in the list box.

top Returns the item number of the item currently at the top of the list box. If there are no items in the list box, an exception is thrown.

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

Win	PM	Motif
Y	Y	Y

Search List

Use these members to search the list box portion of the combination box for a text string. The data structures aid that effort.

locateText Returns the item number of the list box item matching the search string. You can specify the type of search by using the enumeration SearchType (p. 100). If no match is found, this function returns notFound. If there is an error, an exception is thrown.

searchString
String to search for.

caseSensitive
Indicator of whether the search is case-sensitive. The default of true means the search is case-sensitive.

search Use the enumeration SearchType (p. 73) to specify the type of search to perform. The default is exactMatch.

index Index of where to start the search. The default is first.

```
virtual unsigned long
  locateText( const char* searchString,
              Boolean caseSensitive = true,
              SearchType search = exactMatch,
              unsigned long index = first ) const;
```

Win	PM	Motif
Y	Y	Y

Selection

Use these members to set or query the list box's selection state.

deselect Removes the selection state from an item.

index The index of the item.

© IBM Corp. 1992, 1996

```
virtual IBaseListBox&
  deselect( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

deselectAll Removes the selection state from all items in the list box.

```
virtual IBaseListBox&
  deselectAll();
```

Win	PM	Motif
Y	Y	Y

isSelected Returns the selection state of an item. If the item is selected, true is returned. Otherwise, false is returned.

Note: You can use count and isSelected to process all of the selected list box items. However, it is easier to use the cursor facility provided by IBaseListBox::Cursor (p. 101).

```
virtual Boolean
  isSelected( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

numberOfSelections

For single-selection list boxes, if no item is selected, 0 is returned. Otherwise, 1 is returned.

For multiple-selection or extended-selection list boxes, the number of selected items in the list box is returned.

Note: In all cases, the current selection does not change.

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

Win	PM	Motif
Y	Y	Y

select Sets the selection state of an item based on the specified Boolean value.

index Index of the item.

select If you specify true for a list box, based on its selection style, the following occurs:

Single selection The specified item is selected and any previously selected item is deselected.

Multiple selection The specified item is selected.

Extended selection The specified item is selected.

The default is true.

Note: In all cases, if you specify false, the item is deselected.

```
virtual IBaseListBox&
  select( unsigned long index,
          Boolean select = true );
```

Win	PM	Motif
Y	Y	Y

selectAll Sets the selection state for all items in the list box.

```
virtual IBaseListBox&
  selectAll();
```

Win	PM	Motif
Y	Y	Y

selection Returns the 0-based index of the selected item. For single-selection list boxes, the index of the selected item is returned. For multiple-selection or extended-selection list boxes, the index of the *first* selected item is returned.

Note: If no item is selected, the static constant notFound is returned.

```
virtual long
  selection() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the IBaseListBox (p. 81) class. Use these members to set and query list box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 98) unless you have changed the style using setDefaultStyle (p. 94).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

IBaseListBox

disableDrawItem

Disables the style drawItem (p. 98) for the list box.

```
virtual IBaseListBox&
  disableDrawItem();
```

Win	PM	Motif
Y	*Y*	*I*

disableExtendedSelect

Disables the style extendedSelect (p. 98) for the list box.

```
virtual IBaseListBox&
  disableExtendedSelect();
```

Win	PM	Motif
I	*Y*	*Y*

Win An IBaseListBox::Style is used to control this behavior. This platform does not support dynamic change of behavior.

disableMultipleSelect

Disables the style multipleSelect (p. 99) for the list box.

```
virtual IBaseListBox&
  disableMultipleSelect();
```

Win	PM	Motif
I	*Y*	*Y*

Win An IBaseListBox::Style is used to control this behavior. This platform does not support dynamic change of behavior.

disableNoAdjustPosition

Disables the style noAdjustPosition (p. 99) for the list box.

```
virtual IBaseListBox&
  disableNoAdjustPosition();
```

Win	PM	Motif
Y	*Y*	*I*

Motif The AIX release does not process the noAdjustPosition style.

enableDrawItem

Enables or disables the style drawItem (p. 98) for the list box.

```
virtual IBaseListBox&
  enableDrawItem( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*I*

Motif The AIX release does not support this function.

enableExtendedSelect

Enables or disables the style extendedSelect (p. 98) for the list box.

```
virtual IBaseListBox&
  enableExtendedSelect( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

Win An IBaseListBox::Style is used to control this behavior. This platform does not support dynamic change of behavior.

enableMultipleSelect

Enables or disables the style multipleSelect (p. 99) for the list box.

```
virtual IBaseListBox&
  enableMultipleSelect( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

Win An IBaseListBox::Style is used to control this behavior. This platform does not support dynamic change of behavior.

enableNoAdjustPosition

Enables or disables the style noAdjustPosition (p. 99) for the list box.

```
virtual IBaseListBox&
  enableNoAdjustPosition( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*I*

Motif The AIX release does not process the noAdjustPosition style.

isDrawItem

If the style drawItem (p. 98) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isDrawItem() const;
```

Win	PM	Motif
Y	*Y*	*I*

isExtendedSelect

If the style extendedSelect (p. 98) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isExtendedSelect() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isHorizontalScroll

If the style horizontalScroll (p. 99) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isHorizontalScroll() const;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

isMultipleSelect

If the style multipleSelect (p. 99) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isMultipleSelect() const;
```

Win	PM	Motif
Y	Y	Y

isNoAdjustPosition

If the style noAdjustPosition (p. 99) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isNoAdjustPosition() const;
```

Win	PM	Motif
Y	Y	I

Motif The AIX release does not process the noAdjustPosition style. This function always returns false.

setDefaultStyle

Sets the default style for all subsequent list boxes.

style Use the styles provided by IBaseListBox Styles (p. 97) to specify the default list box style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Window Painting

Use these members to display a window.

show

Makes the window visible.

Note: Another window occurring earlier in the Z-order might still obscure this window.

```
virtual IWindow&
  show( Boolean showList = true );
```

Win	PM	Motif
N	N	Y

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup

IControl		
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members provide the implementation of the event-handling mechanism.

passEventToOwner

Determines whether the event is be passed on to the owner.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	*N*	*Y*

registerCallbacks

Registers all possible callbacks and X event handlers to this object for events it might receive. IHandler derived classes later determine which events they will process.

If classes you derive override registerCallbacks, the override must call Inherited::registerCallbacks to register the applicable callbacks and X event handlers.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

© IBM Corp. 1992, 1996

unregisterCallbacks

Unregisters the Motif callbacks for the underlying widget. It is called during destruction of the IBaseListBox window.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Item Changes

Use these members to set or retrieve the number of changes (add or remove items) that have occurred to the list box. These members help track the validity of IBaseListBox::Cursor objects.

changeCount Retrieves the number of changes to the list box.

```
unsigned long
  changeCount() const;
```

Win	PM	Motif
Y	Y	Y

incrementChangeCount

Increments the count of the number of list box changes. All IBaseListBox-derived classes add and remove functions should call this function because add and remove functions may cause an IBaseListBox::Cursor to become invalid.

```
void
  incrementChangeCount();
```

Win	PM	Motif
Y	Y	Y

Layout Support

Use these members to support canvases.

calcMinimumSize

Returns the minimum size of the list box.

The size dimensions are as follows:

- For width, the average character width multiplied by the number of characters set by calling setMinimumCharacters. If you do not call setMinimumCharacters, a default of 25 characters is used.
- For height, the maximum character height multiplied by the minimum number of rows set by calling setMinimumRows. If you do not call setMinimumRows, a default of four rows is used.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enterId Notification identifier provided to observers when the user double-clicks on an item or presses Enter on an item with the cursor in a list box.

```
static INotificationId const
  enterId;
```

Win	PM	Motif
Y	Y	Y

selectId Notification identifier provided to observers when an item is selected in a list box.

```
static INotificationId const
  selectId;
```

Win	PM	Motif
Y	Y	Y

Search List

Use these members to search the list box portion of the combination box for a text string. The data structures aid that effort.

first Searches from the beginning of the string. This is a value for the index parameter of IBaseListBox::locateText.

```
static const unsigned long
  first;
```

Win	PM	Motif
Y	Y	Y

notFound Indicates no match is found. IBaseListBox::locateText returns this value.

```
static const unsigned long
  notFound;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Styles

These style members provide a set of valid styles for the IBaseListBox (p. 81) class. Use these members to set and query list box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

border3D Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	I	I

Win This style is ignored on Windows NT 3.51 with Program Manager.

W32s This style is ignored on Win32s.

classDefaultStyle

Provides the original default style for this class, which is the following: IBaseListBox::horizontalScroll | IBaseListBox::noAdjustPosition | IBaseListBox::border3D | IWindow::visible.

If you do not specify either the extendedSelect or multipleSelect style, the list box is a single-selection list box by default.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

drawItem Draws list box items. It is typically used to display bitmaps.

```
static const Style
  drawItem;
```

Win	PM	Motif
Y	Y	N

Motif The drawItem style is not supported in the AIX environment.

extendedSelect

Extends selection to more than one object. This style is a type of selection optimized for the selection of a single object.

```
static const Style
  extendedSelect;
```

Win	PM	Motif
Y	Y	Y

horizontalScroll

Allows horizontal scrolling of the list box.

```
static const Style
  horizontalScroll;
```

Win	PM	Motif
Y	Y	Y

multipleSelect

Specifies that the user can select any number of objects at a time in the list box or not select any.

```
static const Style
  multipleSelect;
```

Win	PM	Motif
Y	Y	Y

noAdjustPosition

Draws the list box control at the size specified. This can cause an item to be only partially shown.

```
static const Style
  noAdjustPosition;
```

Win	PM	Motif
Y	Y	I

Motif The AIX release does not support the noAdjustPosition style.

Inherited Public Data

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 (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IBaseListBox contains the following nested classes:

IBaseListBox::Cursor (see page 101)
IBaseListBox::Style (see page 105)

SearchType

```
SearchType {
    prefix,
    substring,
    exactMatch
    };
```

Use these enumerators to specify the type of search to perform:

prefix
Matching occurs if the leading characters of the item contain the specified characters.

substring
Matching occurs if the item contains a substring of the specified characters.

exactMatch
Matching occurs if the item is an exact match for the specified characters.

IBaseListBox::Cursor

Derivation

IBase
  IVBase
    IBaseListBox::Cursor

Inherited By None.

Header File ilistbas.hpp

Members

Member	Page	Member	Page
Constructor	101	setToIndex	102
asIndex	103	setToLast	102
Cursor	101	setToNext	103
invalidate	103	setToPrevious	103
isValid	103	~Cursor	102
setToFirst	102		

The nested class IBaseListBox::Cursor defines objects that you can use to traverse through the items in a list box. In the same way that you can use a cursor to traverse through the objects in a collection, you can use a cursor to traverse through a list box, one item at a time.

Once you create a cursor, you can set the cursor to the first, last, or a specific item in a list box. You can add, update, get, and delete list box items based on the cursor's location in the list box.

Note: Before you use the cursor, you must use isValid to verify that the cursor is valid. An exception is thrown if you use a cursor that is not valid to manipulate the list box.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor You can create objects of this class by providing the following:

listbox Reference to a list box control.

© Copyright IBM Corp. 1992, 1996

type A filter type. Use the enumeration Filter (p. 104) to specify this parameter. The default is selectedItems.

Note: When using a selectedItems cursor to navigate through one or more selected items in the list box, note that the cursor is valid for the selection states that exist in the list box at the time the cursor is created. The recommended usage of this cursor is, at some given time, you create the cursor, collect the selection information needed, and then destroy the cursor.

```
Cursor( const IBaseListBox& listbox,
        Filter type = selectedItems );
```

Win	PM	Motif
Y	Y	Y

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Cursor Movement

Use these members to control cursor movement in the list box.

setToFirst Points to the first list box item and validates the cursor. If the cursor is set successfully, true is returned. Otherwise, false is returned.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToIndex Points to the specified item and validates the cursor. If the cursor is set to the specified list box item, true is returned. Otherwise, false is returned.

index A 0-based index indicating the item upon which you want the cursor placed.

```
virtual Boolean
  setToIndex( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

setToLast Points to the last list box item and validates the cursor. If the cursor is set successfully, true is returned. Otherwise, false is returned.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Points to the next item in the list box. If the cursor is set to the next list box item, true is returned. If no more items exist, this function invalidates the cursor and false is returned.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Points to the previous item in the list box. If the cursor is set to the previous list box item, true is returned. If no previous item exists, this function invalidates the cursor and false is returned.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation and Conversion

Use these members to query or return information about the item to which the cursor is pointing.

asIndex Returns the index of the item the cursor is pointing to.

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

Win	PM	Motif
Y	Y	Y

invalidate Flags this cursor as not valid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Returns a Boolean that indicates whether this cursor points to a valid item.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IBaseListBox::Cursor

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Filter

```
Filter {
    selectedItems,
    allItems
    };
```

Use these enumerators to specify which items you want filtered:

selectedItems
Filters only items that are selected.

allItems
Filters all items.

IBaseListBox::Style

Derivation

IBase
  IBitFlag
    IBaseListBox::Style

Inherited By None.

Header File ilistbas.hpp

The nested class IBaseListBox::Style provides a set of valid styles for the IBaseListBox (p. 81) class.

Motif The AIX release does not support the drawItem or the noAdjustPosition style.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IBaseSpinButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
IBaseSpinButton

Inherited By

INumericSpinButton
ITextSpinButton

Header File

ispinbas.hpp

Members

Member	Page	Member	Page
Constructor	115	leftAlign	117
addBorder	108	limit	111
alignment	107	master	117
backgroundColor	109	moveSizeTo	114
border3D	117	noBorder	118
calcMinimumSize	116	pmCompatible	118
centerAlign	117	readOnly	118
convertToGUIStyle	113	registerCallbacks	115
disableDataUpdate	113	removeBorder	108
disableFastSpin	110	resetBackgroundColor	109
enable	112	resetForegroundColor	109
enableDataUpdate	113	rightAlign	118
enableFastSpin	111	servant	118
fastSpin	117	setAlignment	108
foregroundColor	109	setBackgroundColor	109
hasBorder	108	setForegroundColor	110
hasFocus	107	setLimit	111
initialize	115	setMaster	112
isFastSpinEnabled	111	spinDown	112
isMaster	107	spinUp	112
isPMCompatible	107	textId	116
isServant	108	topHandle	110
isSpinFieldValid	111	unregisterCallbacks	115
isWriteable	113	valueId	117
		~IBaseSpinButton	110

IBaseSpinButton is an abstract base class for spin button controls. Use the IBaseSpinButton-derived classes, INumericSpinButton (p. 672), and ITextSpinButton (p. 963), to create and manage numeric and text spin-button controls.

© Copyright IBM Corp. 1992, 1996

Motif Spin buttons created during object construction of IBaseSpinButton-derived classes can use the following Motif widgets:

- An XmForm widget is created with a single line XmText child.
- If the spin button has the style IBaseSpinButton::master, an XmForm widget containing two XmArrowButton widgets is also created as a child of the top XmForm. IWindow::handle returns the handle of the XmText widget.

The User Interface Class Library provides the behavior of an IBaseSpinButton object via private callbacks and a default handler. The IBaseSpinButton-derived classes use a default handler derived from the class IKeyboardHandler. Therefore, attach user-defined handlers derived from IKeyboardHandler to the INumericSpinButton or ITextSpinButton object rather than to its owner window. Doing so enables events to be dispatched to user-defined handlers before the default handler.

Public Functions

Attributes

Use these members to query and change characteristics of the spin button control.

alignment Returns the current alignment in the spin field of this spin button object. The returned value is an enumerator provided by Alignment (p. 119).

```
Alignment
  alignment() const;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This function always returns left.

hasFocus Returns true if the slider has the input focus.

```
virtual Boolean
  hasFocus() const;
```

Win	PM	Motif
Y	*N*	*N*

isMaster If the spin button is a master, true is returned. Otherwise, false is returned.

```
Boolean
  isMaster() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isPMCompatible

Returns true if the control was created with the pmCompatible (p. 118) style.

© IBM Corp. 1992, 1996

```
Boolean
  isPMCompatible() const;
```

Win	PM	Motif
Y	Y	Y

isServant If the spin button is a servant, true is returned. Otherwise, false is returned.

```
Boolean
  isServant() const;
```

Win	PM	Motif
Y	Y	Y

setAlignment Changes the text or number alignment in the spin field.

alignment Use the enumeration Alignment (p. 119) to specify the alignment of the text or number. The default is left.

```
virtual IBaseSpinButton&
  setAlignment( Alignment alignment = left );
```

Win	PM	Motif
I	Y	I

Win The alignment cannot be changed after the control is created. Use the styles leftAlign (p. 117), centerAlign (p. 117) or rightAlign (p. 118) on the constructor to specify alignment.

Borders

Use these members to query and change the border style of the spin button object. The border style determines if the spin button control is drawn with a border around the spin field.

addBorder Adds a border to the spin button.

```
virtual IBaseSpinButton&
  addBorder( Boolean add = true );
```

Win	PM	Motif
Y	Y	Y

hasBorder If the spin button has a border, true is returned. Otherwise, false is returned.

```
Boolean
  hasBorder() const;
```

Win	PM	Motif
Y	Y	Y

removeBorder

Removes the border from the spin button.

```
virtual IBaseSpinButton&
  removeBorder();
```

Win	PM	Motif
Y	Y	Y

Colors

Use these members to query, set, and reset colors for IBaseSpinButton and derived classes.

backgroundColor

Returns the background color value of the spin button area. If you have not set a color for the area, the default is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	N	N

This member is overridden in this derived class for specific operating system behavior.

foregroundColor

Returns the foreground color value of the spin button area. If you have not set a color for the area, the default is returned.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	Y	N

This member is overridden in this derived class for specific operating system behavior.

resetBackgroundColor

Resets the background color by undoing a previous set.

```
virtual IBaseSpinButton&
  resetBackgroundColor();
```

Win	PM	Motif
Y	N	N

This member is overridden in this derived class for specific operating system behavior.

resetForegroundColor

Resets the foreground color by undoing a previous set.

```
virtual IBaseSpinButton&
  resetForegroundColor();
```

Win	PM	Motif
Y	N	N

This member is overridden in this derived class for specific operating system behavior.

setBackgroundColor

Sets the background color to the indicated color.

© IBM Corp. 1992, 1996

IBaseSpinButton

```
virtual IBaseSpinButton&
  setBackgroundColor( const IColor& color );
```

Win	PM	Motif
Y	N	N

This member is overridden in this derived class for specific operating system behavior.

setForegroundColor

Sets the foreground color to the indicated color.

```
virtual IBaseSpinButton&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
Y	N	N

This member is overridden in this derived class for specific operating system behavior.

Compound Control

Use these members to access the top window system.

topHandle

Returns the topmost manager widget of the spin button.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	N	Y

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor provided for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

~IBaseSpinButton

```
virtual
  ~IBaseSpinButton();
```

Win	PM	Motif
Y	Y	Y

Fast Spin

Use these members to query and change the fastSpin style of the spin button object. The fastSpin style specifies the spin speed of the spin button is increased with time. The speed doubles every two seconds.

disableFastSpin

Disables fast spinning of the spin button.

```
virtual IBaseSpinButton&
  disableFastSpin();
```

Win	PM	Motif
Y	Y	Y

enableFastSpin

Enables or disables fast spinning of the spin button. The style fastSpin causes the spin button to increase the spin speed relative to the length of time the button is pressed. The speed doubles every two seconds until the spin button reaches the upper or lower bound. At this point, the speed resets to the original speed and again doubles every two seconds.

Note: Do not use this function on a master spin button that has servants spun from it.

```
virtual IBaseSpinButton&
  enableFastSpin( Boolean fast = true );
```

Win	PM	Motif
Y	Y	Y

isFastSpinEnabled

If the spin speed is doubled every two seconds, true is returned. Otherwise, false is returned.

```
Boolean
  isFastSpinEnabled() const;
```

Win	PM	Motif
Y	Y	Y

Limit and Spin

Use these members to manage the spin field for objects of this class.

isSpinFieldValid

Determines if the contents of the spin field are valid. It is a pure virtual function that derived classes use.

```
virtual Boolean
  isSpinFieldValid( Boolean caseSensitive ) const = 0;
```

Win	PM	Motif
Y	Y	Y

limit

Returns the number of characters permitted in the spin field.

```
unsigned long
  limit() const;
```

Win	PM	Motif
Y	Y	Y

setLimit

Sets the number of characters permitted in the spin field. The User Interface Class Library defines this limit as 255 at the time of construction.

© IBM Corp. 1992, 1996

```
virtual IBaseSpinButton&
  setLimit( unsigned long aNumber = 255 );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The specified limit value is invalid.

setMaster Defines a servant spin button's master. The spin button for which you call this function must be a servant.

```
virtual IBaseSpinButton&
  setMaster( IBaseSpinButton& master );
```

Win	PM	Motif
Y	Y	Y

Win The presence or absence of the pmCompatible style must be the same for a master spin button and its associated servant spin buttons.

Exceptions	
IInvalidRequest.	This spin button is not a servant.
IInvalidParameter	The specified spin button is not a master spin button.
IAccessError	The operating system's request to set the master spin button has failed.

spinDown Spins the button down the specified number of times.

```
virtual IBaseSpinButton&
  spinDown( unsigned long spinBy = 1 );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The spin value is invalid.

spinUp Spins the button up the specified number of times.

```
virtual IBaseSpinButton&
  spinUp( unsigned long spinBy = 1 );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The spin value is invalid.

Overrides

These members have been overridden to provide the correct behavior for this class.

enable Enables or disables the window from accepting keyboard and mouse input.

```
virtual IBaseSpinButton&
  enable( Boolean enable = true );
```

Win	PM	Motif
Y	Y	N

Read-Only Operations

These members query and modify the read-only mode in the spin button, which specifies if the user can directly change the spin field text.

disableDataUpdate

Disables direct editing of the spin field data by the end user.

```
virtual IBaseSpinButton&
  disableDataUpdate();
```

Win	PM	Motif
Y	Y	Y

enableDataUpdate

Enables or disables direct editing of spin field data by the end user.

```
virtual IBaseSpinButton&
  enableDataUpdate( Boolean writeable = true );
```

Win	PM	Motif
Y	Y	Y

isWriteable

If the user can type in the spin field, true is returned. Otherwise, false is returned.

```
Boolean
  isWriteable() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the IBaseSpinButton (p. 106) class. Use these members to query and set spin button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

© IBM Corp. 1992, 1996

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	*Y*	*N*

Window Positioning

These members have been overridden to provide the correct behavior for this class.

moveSizeTo Changes the position and size of the spin button window using a coordinate system that is lower-left origin-based.

rectangle A rectangle for the spin button window.

```
virtual IBaseSpinButton&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

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 (p. 1044) are not shown.

Protected Functions

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor provided for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

IBaseSpinButton

Derived classes use this protected constructor to construct objects of this class. This is the default constructor and accepts no parameters.

```
IBaseSpinButton();
```

Win	PM	Motif
Y	Y	Y

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Registers Motif callbacks for the widgets created during construction of objects of derived classes.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

unregisterCallbacks

Unregisters Motif callbacks for the widgets created during construction of objects of derived classes.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Implementation

These members provide utilities used to implement this class.

initialize

Used by several of the constructors to create a spin button control.

1
```
virtual void
  initialize();
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

IBaseSpinButton

2 `void initialize( unsigned long windowId, const IWindowHandle& parent, const IWindowHandle& owner, unsigned long style, const IRectangle& initial );`

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the recommended minimum size of this spin button control. The size is based on the font and the text limit.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify notifications sent to observer objects.

textId Notification identifier provided to observers when the text value of a spin button changes.

```
static INotificationId const
  textId;
```

Win	PM	Motif
Y	Y	Y

valueId Notification identifier provided to observers when the value of a numeric spin button changes.

```
static INotificationId const
  valueId;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for the IBaseSpinButton (p. 106) class. Use these members to query and set spin button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

border3D Adds an etched 3D border to the control. This style is ignored if the pmCompatible or noBorder styles are chosen.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	*I*	*I*

Win This style is ignored on Windows NT 3.51 with Program Manager.

W32s This style is ignored on Win32s.

centerAlign Centers the text in the spin field.

```
static const Style
  centerAlign;
```

Win	PM	Motif
Y	*Y*	*I*

fastSpin Increases the spin speed of the spin button with time. The speed doubles every two seconds.

```
static const Style
  fastSpin;
```

Win	PM	Motif
Y	*Y*	*Y*

leftAlign Left-justifies the text in the spin field.

```
static const Style
  leftAlign;
```

Win	PM	Motif
Y	*Y*	*I*

master Specifies that the spin button component consists of the following:

- At least one single-line entry field (SLE) or spin field
- Two arrows, the Up arrow and the Down arrow

© IBM Corp. 1992, 1996

When the spin button contains more than one spin field, the master component contains the spin arrows. If the component contains only one spin field, it must be a master.

```
static const Style
  master;
```

Win	PM	Motif
Y	Y	Y

noBorder Suppresses the drawing of a border.

```
static const Style
  noBorder;
```

Win	PM	Motif
Y	Y	Y

pmCompatible

Creates a control with a Presentation Manager look and feel.

```
static const Style
  pmCompatible;
```

Win	PM	Motif
Y	Y	Y

PM This style is always set.

Win Specify this style to create a control with a look and feel similar to the Presentation Manager spin button control.

The presence or absence of the pmCompatible style must be the same for a master spin button and its associated servant spin buttons.

readOnly Prevents entering input into the spin field.

```
static const Style
  readOnly;
```

Win	PM	Motif
Y	Y	Y

rightAlign Right-justifies the text in the spin field.

```
static const Style
  rightAlign;
```

Win	PM	Motif
Y	Y	I

servant Creates a multiple-field spin button by spinning servants from the master. This style requires that you call ISpinButton::setMaster to define the master spin field.

```
static const Style
  servant;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IBaseSpinButton contains the following nested classes:

IBaseSpinButton::Style (see page 121)

Alignment

```
Alignment {
    left,
    right,
    center
    };
```

Use these enumerators to specify the alignment of text in the spin field:

left

Left-justifies the text or number.

© IBM Corp. 1992, 1996

center
Centers the text or number.

right
Right-justifies the text or number.

Motif The AIX release ignores the Alignment enumeration. All spin buttons have left alignment.

IBaseSpinButton::Style

Derivation

IBase
  IBitFlag
    IBaseSpinButton::Style

Inherited By None.

Header File ispinbas.hpp

The nested class IBaseSpinButton::Style provides a set of valid styles for the IBaseSpinButton (p. 106) class.

The AIX release does not support the centerAlign and rightAlign styles. Spin button text and numbers are left-aligned in AIX.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

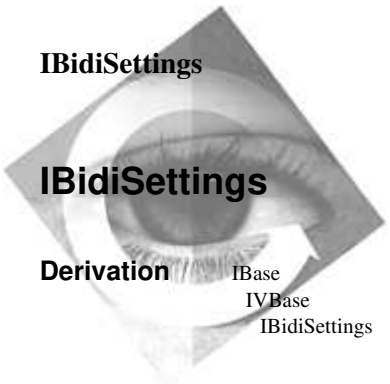
Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



IBidiSettings

Derivation

IBase
IVBase
IBidiSettings

Inherited By None.

Header File ibidiset.hpp

Members

Member	Page	Member	Page
Constructor	127	setNumeralDisplay	124
apply	126	setTextOrientation	124
disableSymmetricSwapping	122	setTextShape	124
disableWordByWordReordering	123	setTextType	124
enableSymmetricSwapping	123	setWindowLayout	124
enableWordByWordReordering	123	textOrientation	125
isBidiSupported	126	textShape	125
isSymmetricSwappingEnabled	123	textType	125
isWordByWordReorderingEnabled	123	windowLayout	125
numeralDisplay	124	~IBidiSettings	127

The IBidiSettings class identifies information about the bidirectional national language support for a window or graphic context. Objects of this class are created to query or set the bidirectional attributes of a window or a graphic context. An IBidiSettings object identifies the bidirectional attributes of a window or graphic context at one point in time and is not updated if the bidirectional attributes of the window or graphic context change. Changes to the IBidiSettings object are not reflected in the window until IBidiSettings::apply is used.

Public Functions

Attributes

Use these members to query and set the bidirectional attributes.

disableSymmetricSwapping

Disables the swapping of directional characters.

```
virtual IBidiSettings&
  disableSymmetricSwapping();
```

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

disableWordByWordReordering

Disables word-by-word reordering.

```
virtual IBidiSettings&
  disableWordByWordReordering();
```

Win	PM	Motif
Y	*Y*	*N*

enableSymmetricSwapping

Enables the swapping of directional characters.

Directional characters are characters, such as parentheses, brackets, and braces. In left-to-right text, an "open" bracket is a left bracket, while for right-to-left text an "open" bracket is a right bracket.

When symmetric swapping is enabled, the system automatically swaps between the symmetrical characters if right-to-left text orientation is used.

```
virtual IBidiSettings&
  enableSymmetricSwapping( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*N*

enableWordByWordReordering

Enables word-by-word reordering.

When word-by-word reordering is enabled, each word is evaluated for text reordering. Text reordering includes the reversal of text for display and the swapping of directional characters when using right-to-left text orientation.

When word-by-word reordering is disabled, only the first word is evaluated for text reordering, and the entire text uses the result.

```
virtual IBidiSettings&
  enableWordByWordReordering( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*N*

isSymmetricSwappingEnabled

If symmetric swapping is enabled, true is returned. Otherwise, false is returned.

```
Boolean
  isSymmetricSwappingEnabled() const;
```

Win	PM	Motif
Y	*Y*	*N*

isWordByWordReorderingEnabled

If word-by-word reordering is enabled, true is returned. Otherwise, false is returned.

© IBM Corp. 1992, 1996

```
Boolean
  isWordByWordReorderingEnabled() const;
```

Win	PM	Motif
Y	Y	N

numeralDisplay

Returns the bidirectional attribute that specifies how numerals (digits) are processed for display.

```
BidiNumeralType
  numeralDisplay() const;
```

Win	PM	Motif
Y	Y	N

setNumeralDisplay

Sets the bidirectional attribute that specifies how numerals (digits) are processed for display.

```
virtual IBidiSettings&
  setNumeralDisplay( BidiNumeralType numeralDisplay );
```

Win	PM	Motif
Y	Y	N

setTextOrientation

Sets the bidirectional attribute that specifies how bidirectional text is oriented for display.

```
virtual IBidiSettings&
  setTextOrientation( BidiTextOrientation textOrientation );
```

Win	PM	Motif
Y	Y	N

setTextShape

Sets the bidirectional attribute that specifies how Arabic text is processed for display and how new text typed by the user is stored.

```
virtual IBidiSettings&
  setTextShape( BidiTextShape textShape );
```

Win	PM	Motif
Y	Y	N

setTextType

Sets the bidirectional attribute that specifies how bidirectional text is detected and processed.

```
virtual IBidiSettings&
  setTextType( BidiTextType textType );
```

Win	PM	Motif
Y	Y	N

setWindowLayout

Sets the bidirectional attribute that specifies how objects in a window or dialog are positioned and justified.

```
virtual IBidiSettings&
  setWindowLayout( BidiLayout windowLayout );
```

Win	PM	Motif
Y	Y	N

textOrientation

Returns the bidirectional attribute that specifies how bidirectional text is oriented for display.

```
BidiTextOrientation
  textOrientation() const;
```

Win	PM	Motif
Y	Y	N

textShape

Returns the bidirectional attribute that specifies how Arabic text is processed for display and how new text typed by the user is stored.

```
BidiTextShape
  textShape() const;
```

Win	PM	Motif
Y	Y	N

textType

Returns the bidirectional attribute that specifies how bidirectional text is detected and processed for display.

```
BidiTextType
  textType() const;
```

Win	PM	Motif
Y	Y	N

windowLayout

Returns the bidirectional attribute that specifies how objects in the window or dialog are positioned and justified.

```
BidiLayout
  windowLayout() const;
```

Win	PM	Motif
Y	Y	N

Bidirectional Support

Use these members to query bidirectional support and set bidirectional attributes for a window or graphic context.

Reading and writing of Arabic or Hebrew text is done from right to left. Thus, the starting point of an Arabic or Hebrew text is on the right side of the page or screen while the ending of the text is on the left. Arabic and Hebrew text can also contain embedded text that is written and read from left to right (such as the use of numbers or embedded Latin text). Having left-to-right text embedded in right-to-left text is the reason we refer to Arabic and Hebrew languages as bidirectional languages.

© IBM Corp. 1992, 1996

IBidiSettings

Applications that are developed for Arabic and Hebrew languages must accommodate bidirectional data. Bidirectional attributes query and set information about how text is stored and displayed on Arabic or Hebrew systems.

apply

Changes the bidirectional attributes.

1

```
virtual IBidiSettings&
  apply( const IWindow& window,
         Boolean childInherit = true,
         Boolean refresh = true );
```

Win	PM	Motif
Y	Y	N

Changes the bidirectional attributes of the window to the values defined in the IBidiSettings object. If *childInherit* is set to true, the bidirectional attributes of all child windows are also changed. If *refresh* is set to true, the window is invalidated, and the screen is updated to reflect the new bidirectional attributes.

2

```
virtual IBidiSettings&
  apply( const IWindowHandle windowHandle,
         Boolean childInherit = true,
         Boolean refresh = true );
```

Win	PM	Motif
Y	Y	N

Changes the bidirectional attributes of the window to the values defined in the IBidiSettings object. If *childInherit* is set to true, the bidirectional attributes of all child windows are also changed. If *refresh* is set to true, the window is invalidated, and the screen is updated to reflect the new bidirectional attributes.

3

```
virtual IBidiSettings&
  apply( const IGraphicContext& graphicContext );
```

Win	PM	Motif
Y	Y	N

Changes the bidirectional attributes of the graphic context to the values defined in the IBidiSettings object.

4

```
virtual IBidiSettings&
  apply( const IPresSpaceHandle presSpace );
```

Win	PM	Motif
Y	Y	N

Changes the bidirectional attributes of the presentation space to the values defined in the IBidiSettings object.

isBidiSupported

If the operating system supports querying and setting of bidirectional attributes, true is returned. Otherwise, false is returned.

```
static Boolean
  isBidiSupported();
```

Win	PM	Motif
Y	Y	N

Constructors

You can construct and delete objects of the IBidiSettings class. You cannot copy or assign IBidiSettings objects because both the copy constructor and the assignment operator are private functions.

IBidiSettings

1 `IBidiSettings( const IWindowHandle windowHandle );`

Win	PM	Motif
Y	Y	N

Constructs objects of the IBidiSettings class. The resulting object contains the current bidirectional attributes for the specified window.

2 `IBidiSettings( const IWindow& window );`

Win	PM	Motif
Y	Y	N

Constructs objects of the IBidiSettings class. The resulting object contains the current bidirectional attributes for the specified window.

3 `IBidiSettings( const IGraphicContext& graphicContext );`

Win	PM	Motif
Y	Y	N

Constructs objects of the IBidiSettings class. The resulting object contains the current bidirectional attributes for the specified graphic context.

4 `IBidiSettings( const IPresSpaceHandle presSpace );`

Win	PM	Motif
Y	Y	N

Constructs objects of the IBidiSettings class. The resulting object contains the current bidirectional attributes for the specified presentation space.

~IBidiSettings

```
virtual
  ~IBidiSettings();
```

Win	PM	Motif
Y	Y	N

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBidiSettings

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

BidiLayout

```
BidiLayout {
    layoutLeftToRight,
    layoutRightToLeft
    };
```

Use these enumerations to specify how objects within a window or dialog are positioned (layout) and how objects are justified:

layoutLeftToRight
Objects within a window or dialog are positioned with the origin at the bottom-left of the window, and the default justification is on the left side of the object.

layoutRightToLeft
Objects within a window or dialog are positioned with the origin at the bottom-right of the window, and the default justification is on the right side of the object.

BidiNumeralType

```
BidiNumeralType {
    arabic,
    asStored,
    national,
    contextual
    };
```

Use these enumerations to specify how numerals (digits) are processed for display:

arabic
Only Arabic numerals are displayed.

asStored
Numerals are displayed in the order that they are stored in memory.

national
Only national (Hindi) numerals are displayed.

contextual
Numerals are displayed according to surrounding text.

BidiTextOrientation

```
BidiTextOrientation {
    textLeftToRight,
    textRightToLeft,
    textContextual
    };
```

Use these enumerations to specify how bidirectional text is oriented for display:

textLeftToRight
Text that has a BidiTextType of visual is displayed as it is stored (text starts at the left and ends at the right). Text that has a BidiTextType of implicit is displayed so that Latin text retains its order, while Arabic or Hebrew text is reordered (reversed) for display.

textRightToLeft
Text that has a BidiTextType of visual is displayed in the reverse order from how it is stored (text starts at the right and ends at the left). Text that has a BidiTextType of implicit is displayed so that Arabic or Hebrew text retains its order, while Latin text is reordered (reversed) for display.

textContextual
The orientation of the text for display is determined by the first strong (nonneutral) character in the text. If the first character is an English character, the orientation is left to right. If the first character is an Arabic or Hebrew character, the orientation is right to left. If the first character is a neutral character (such as a space or punctuation mark), then the orientation is determined by the next character in the storage buffer.

BidiTextShape

```
BidiTextShape {
    displayShaped, saveShaped,    nominalShape,  initialShape,  middleShape,
    finalShape,    isolatedShape
    };
```

Use these enumerations to specify how Arabic text is processed for display and how new text typed by the user is stored.

displayShaped
Text is stored in nominal shapes (one codepoint per letter) and is shaped upon display. New text typed by the user is stored in nominal shapes (one codepoint per letter).

© IBM Corp. 1992, 1996

IBidiSettings

saveShaped
Text is already shaped and is not shaped upon display. New text typed by the user is shaped by the control before being stored or displayed. This mode is valid only when the BidiTextType is visual.

nominalShape
Text is stored in nominal shapes (one codepoint per letter) and is not shaped upon display. New text typed by the user is stored in nominal shapes. This mode is not commonly used and is supported to be compatible with a "shaping-off" state used on host platforms.

initialShape
Text is already shaped and is not shaped upon display. New text typed by the user is shaped as if it came at the beginning of a word before being stored or displayed. This mode is valid only when the BidiTextType is visual.

middleShape
Text is already shaped and is not shaped upon display. New text typed by the user is shaped as if it came in the middle of a word before being stored or displayed. This mode is valid only when the BidiTextType is visual.

finalShape
Text is already shaped and is not shaped upon display. New text typed by the user is shaped as if it came at the end of a word before being stored or displayed. This mode is valid only when the BidiTextType is visual.

isolatedShape
Text is already shaped and is not shaped upon display. New text typed by the user is shaped as if isolated (a single character word) before being stored or displayed. This mode is valid only when the BidiTextType is visual.

BidiTextType

```
BidiTextType {
    visual,
    implicit
    };
```

Use these enumerations to specify how bidirectional text is detected and processed:

visual
The whole text has the same direction. Whether it is left to right or right to left is determined by the text orientation attribute. Whether the text contains digits, English, Arabic, or Hebrew characters does not change the sequence of characters within the given text.

implicit
The determination of text direction is based on the actual sequence of characters in the text.

IBitmapHandle

Derivation

IBase
  IHandle
    IBitmapHandle

Inherited By

ISystemBitmapHandle

Header File

ihandle.hpp

Members

Member	Page	Member	Page
Constructor	131	operator Value	132
operator =	132	~IBitmapHandle	132

The IBitmapHandle class accesses and manages bitmap resources through reference-counting. Reference-counting allows the system to use one bitmap in multiple places, and the library maintains the lifetime of this bitmap until all users are finished with it.

PM IBitmapHandle is an alias for the OS/2 Programmer's Toolkit type HBITMAP.

Motif IBitmapHandle is an alias for the XLib Pixmap type.

Public Functions

Constructors

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

The library provides the copy constructor, destructor, and assignment operator to keep track of references to the bitmap so its resources can be freed when no longer needed.

IBitmapHandle

1 `IBitmapHandle( Value value = 0 );`

Win	PM	Motif
Y	Y	Y

You can construct an object of this class from a bitmap handle (a value of type IHandle::Value), which defaults to 0.

© Copyright IBM Corp. 1992, 1996

IBitmapHandle

2 `IBitmapHandle( const IBitmapHandle& aHandle );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class from an existing IBitmapHandle object.

operator = Assigns the value of one bitmap handle to another while managing the references to the two handles. If the reference count of the bitmap previously associated with the target handle reaches 0, the bitmap is destroyed.

```
IBitmapHandle&
  operator =( const IBitmapHandle& aHandle );
```

Win	PM	Motif
Y	Y	Y

~IBitmapHandle

```
~IBitmapHandle();
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
N	N	Y

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

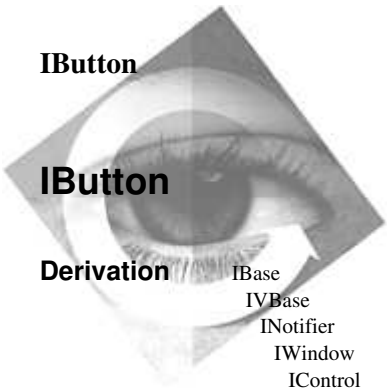
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton

Inherited By

ICustomButton
IPushButton
ISettingButton

Header File ibutton.hpp

Members

Member	Page	Member	Page
Constructor	139	foregroundColor	135
allowsMouseClickFocus	136	highlight	136
backgroundColor	134	hiliteBackgroundColor	135
buttonClickId	139	hiliteForegroundColor	135
click	137	isHighlighted	136
disabledForegroundColor	135	noPointerFocus	140
disableMouseClickFocus	137	setText	138
enableMouseClickFocus	137	unhighlight	136
enableNotification	137	~IButton	136

The IButton class is the base class for the button control classes. This class contains the common functions for all button controls. You can construct button objects by using classes derived from IButton.

Public Functions

Colors

Use these members to query or set the color used for various areas of a window.

backgroundColor

Returns the background color value of the window area. If you have not set the color for the area, the default color is returned.

© Copyright IBM Corp. 1992, 1996

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	*Y*	*N*

disabledForegroundColor

Returns the disabled foreground color value of the window area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  disabledForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*N*

Win Returns the default disabled foreground color value of the window area.

foregroundColor

Returns the foreground color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	*Y*	*N*

hiliteBackgroundColor

Returns the highlight background color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  hiliteBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*N*

Win Returns the default highlight background color value of the window area.

hiliteForegroundColor

Returns the highlight foreground color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  hiliteForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*N*

Win Returns the default highlight foreground color value of the window area.

Constructors

The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. You can destruct objects of this class.

© IBM Corp. 1992, 1996

IButton

~IButton

```
virtual
  ~IButton();
```

Win	PM	Motif
Y	Y	Y

Highlighting

Use these members to query and modify the highlight state of the button. A button in the highlighted state has the same appearance as if the mouse selection button (mouse button 1) was pressed while the mouse pointer was over the button control.

highlight Sets the button's highlight state.

```
virtual IButton&
  highlight( Boolean highlight = true );
```

Win	PM	Motif
Y	Y	I

isHighlighted If the button's highlight state is set, true is returned. Otherwise, false is returned.

```
Boolean
  isHighlighted() const;
```

Win	PM	Motif
Y	Y	I

Motif This function always returns false.

unhighlight Turns off the button's highlight state.

```
virtual IButton&
  unhighlight();
```

Win	PM	Motif
Y	Y	I

Mouse Focus

Mouse focus describes a button's capability to receive the input focus when the mouse is clicked with the pointer over the button. Use these members to query and modify this capability of a button object.

allowsMouseClickFocus

If the button can receive the focus when clicked with the mouse, true is returned. Otherwise, false is returned.

```
Boolean
  allowsMouseClickFocus() const;
```

Win	PM	Motif
Y	Y	I

Motif This function always returns true.

disableMouseClickFocus

Prevents the button from receiving the focus when the user selects it using the mouse.

```
virtual IButton&
  disableMouseClickFocus();
```

Win	PM	Motif
Y	Y	I

enableMouseClickFocus

Enables or disables the button to receive the focus when the user selects it using the mouse.

enable Flag indicating whether the mouse can have focus.

```
virtual IButton&
  enableMouseClickFocus( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

Motif In the AIX version of User Interface Class Library, buttons always receive the focus when clicked on.

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables or disables the button to send notifications to any observer objects.

```
virtual IButton&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Selection

These members change the selection state of the button. The selection state of a button is typically changed by the user clicking on it using the mouse or pressing a key to select it. You can also change or query the state using other User Interface Class Library functions.

click

Simulates the user clicking on the button control using the mouse selection button. The click function calls ISelectHandler. You might use this function instead of ISettingButton::select, which does not call ISelectHandler.

```
virtual IButton&
  click();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Text

Use these functions to set the button text. *Button text* is the label string placed on or beside the button.

setText Sets the text for the button. If appropriate, this function notifies a parent canvas to update the layout for its children.

text Reference to the text object or text for the button.

1
```
virtual IButton&
  setText( const IResourceId& text );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual IButton&
  setText( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Constructors

The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. You can destruct objects of this class.

IButton

```
IButton();
```

Win	PM	Motif
Y	*Y*	*Y*

Derived classes use this protected constructor to create objects of this class.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

buttonClickId Notification identifier provided to observers when the button control is clicked by the user.

```
static INotificationId const
  buttonClickId;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for use in classes derived from IButton. Use these members to query and set button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

© IBM Corp. 1992, 1996

noPointerFocus

Enables the cursor to stay on a control for which information is required, rather than moving to the button clicked on by the user. A typical usage is for a help push button that shows contextual help.

Note: This has no effect on keyboard interaction.

```
static const Style
  noPointerFocus;
```

Win	PM	Motif
Y	Y	Y

Motif Motif does not support the noPointerFocus style.

Inherited Public Data

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IButton contains the following nested classes:

IButton::Style (see page 142)

© IBM Corp. 1992, 1996

IButton::Style

Derivation

IBase
IBitFlag
IButton::Style

Inherited By None.

Header File ibutton.hpp

The nested class IButton::Style provides a set of valid styles for the IButton (p. 134) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IButtonNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          IButtonNotifyHandler

Inherited By ISettingButtonNotifyHandler

Header File ibtnnhdr.hpp

Members

Member	Page
Constructor	143
dispatchHandlerEvent	144
~IButtonNotifyHandler	143

The IButtonNotifyHandler class processes events for all classes of buttons.

This class is designed to handle events that require the button class to generate a notification. If notifications are enabled for this class, a notification will be generated and sent to all observers when the proper conditions for the specific notification exist.

Public Functions

Constructors

You can construct and destruct objects of this class.

IButtonNotifyHandler

This is the default constructor and accepts no parameters.

```
IButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IButtonNotifyHandler

© Copyright IBM Corp. 1992, 1996

IButtonNotifyHandler

```
virtual
  ~IButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the button observers if the following event is received:

- buttonClicked event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

ITextControlNotifyHandler		
dispatchHandlerEvent		

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ICheckBox

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
ISettingButton
ICheckBox

Inherited By None.

Header File icheckbx.hpp

Members

Member	Page	Member	Page
Constructor	148	enableAutoSelect	147
autoSelect	152	isAutoSelect	147
calcMinimumSize	151	registerCallbacks	150
classDefaultStyle	152	setDefaultStyle	149
convertToGUIStyle	149	unregisterCallbacks	151
defaultStyle	149	~ICheckBox	148
disableAutoSelect	147		

The ICheckBox class creates and manages check box control windows.

A *check box* is a square box with associated text that represents a choice. When a user selects the choice, the check box is filled to indicate that the choice is selected. If the user selects the check box again, the choice is deselected.

Process selection of a check box by deriving objects from the ISelectHandler (p. 851) class and by adding a handler to either the check box object or to its owner window.

You can attach the following handlers to this control:

- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISelectHandler (p. 851)

© Copyright IBM Corp. 1992, 1996

Public Functions

Auto Select

Auto select members query and modify the autoSelect style of a check box object. The autoSelect style determines whether the state of the check box is automatically changed when the user clicks on it.

disableAutoSelect

Removes the autoSelect (p. 152) style from the check box control. If auto select is disabled, the application is responsible for changing the state of the check box when the check box is clicked.

```
virtual ICheckBox&
  disableAutoSelect();
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function; therefore, it has no effect.

enableAutoSelect

Sets the check box control to the style autoSelect (p. 152) or removes the autoSelect style from the control.

```
virtual ICheckBox&
  enableAutoSelect( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Motif The AIX version does not support this function; therefore, it has no effect. In the AIX version, check boxes are always automatically selected.

isAutoSelect If the check box control has the autoSelect (p. 152) style set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isAutoSelect() const;
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function; therefore, it returns true. In the AIX version, check boxes are always automatically selected.

Constructors

You can construct and destruct objects of the ICheckBox class. You cannot copy or assign ICheckBox objects because both the copy constructor and assignment operator are private functions.

© IBM Corp. 1992, 1996

ICheckBox

ICheckBox

1
```
ICheckBox( unsigned long id,
           IWindow* parent,
           IWindow* owner,
           const IRectangle& initial = IRectangle ( ),
           const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Constructs a check box control and an object for it using the following parameters:

id The window ID of the check box.

parent The parent window.

owner The owner window.

initial The initial position and size of the check box you are constructing. The default is IRectangle (Vol. I). Optional.

style The check box's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2
```
ICheckBox( unsigned long id,
           IWindow* parent );
```

Win	PM	Motif
Y	Y	Y

Constructs an object from a dialog (parent) window, and the ID of a check box control on that dialog window.

id The window ID of the check box.
parent The dialog (parent) window.

3
```
ICheckBox( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

Constructs the object using the handle of an existing check box window.

handle The window handle of an existing check box control.

~ICheckBox

```
virtual
  ~ICheckBox();
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set check box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 152) unless you have changed the style using setDefaultStyle (p. 149).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent check boxes.

style Use the styles provided by ICheckBox Styles (p. 151) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

ISettingButton		
deselect	enableAutoSelect	isAutoSelect
disableAutoSelect	enableNotification	isSelected

© IBM Corp. 1992, 1996

ICheckBox

IButton		
allowsMouseClickFocus	disableMouseClickFocus	highlight
backgroundColor	enableMouseClickFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Registers Motif callbacks for the widgets created during object construction of this class.

Registers all possible callbacks and X event handlers to this object for events it may receive. IHandler derived classes later determine which events they will process.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

unregisterCallbacks

Unregisters Motif callbacks for the widget created during object construction of this class.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size this check box control can be, based on the text string length and the current font.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

ISettingButton		
passEventToOwner		

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

© IBM Corp. 1992, 1996

Styles

These style members provide a set of valid styles for this class. Use these members to query and set check box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

autoSelect The auto select style determines whether the state of the check box is automatically changed when the user clicks on it. If auto select is enabled, the state of the check box automatically cycles between the selected and not selected states. If auto select is disabled, the application must change the state of the check box when the check box is clicked.

```
static const Style
  autoSelect;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The autoSelect style is always selected in the Motif version. You cannot disable auto selection of check boxes.

classDefaultStyle

Provides the original default style for this class, which is the following: ICheckBox::autoSelect | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

ISettingButton		
selectId		

IButton		
buttonClickId	**noPointerFocus**	

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ICheckBox contains the following nested classes:

ICheckBox::Style (see page 154)

© IBM Corp. 1992, 1996

ICheckBox::Style

Derivation IBase
IBitFlag
ICheckBox::Style

Inherited By None.

Header File icheckbx.hpp

The nested class ICheckBox::Style provides a set of valid styles for the ICheckBox (p. 146) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IClipboard

Derivation

IBase
  IVBase
    IClipboard

Inherited By None.

Header File iclipbrd.hpp

Members

Member	Page	Member	Page
Constructor	163	isOpen	162
bitmap	156	metafileFormat	165
bitmapFormat	164	open	162
close	161	owner	161
data	157	paletteFormat	165
displayBitmapFormat	164	primaryFormat	160
displayMetafileFormat	165	registerFormat	161
displayTextFormat	165	setBitmap	157
empty	162	setData	158
format	159	setHandle	158
formatAsHandle	160	setOwner	164
formatCount	160	setText	159
hasBitmap	157	text	159
hasData	157	textFormat	166
hasText	157	~IClipboard	163

The IClipboard class provides support for writing data to and reading data from the system clipboard. While you can only store a single item of data in the clipboard, you can store this item in multiple formats. IClipboard predefines several system clipboard formats. In addition, any application can create and register additional private formats.

Before you can write any data to, or read any data from, the clipboard, you first open it. Only a single application at a time can open the clipboard. If an application tries to open the clipboard but another application already has it open, it waits until the clipboard is available. Therefore, the default behavior of the clipboard classes minimizes the time the clipboard is open.

If you use the default behavior of IClipboard, the clipboard functions that require an open clipboard will open it when needed and close it when finished. You turn off the default behavior of IClipboard when you explicitly open the clipboard by calling IClipboard::open (p. 162). If you open the clipboard in this manner, IClipboard functions will not close the clipboard. If you explicitly open the clipboard, close the

© Copyright IBM Corp. 1992, 1996

clipboard by calling IClipboard::close (p. 161). You can turn off the default behavior of IClipboard to place several different formats of your data on the clipboard without opening and closing it to write each format.

All clipboard operations must be associated with a window. You provide this window on the IClipboard constructor. If necessary, IClipboard makes this window the owner of the clipboard. The clipboard owner is the window responsible for the data put on the clipboard. It is also the window that the operating system sends messages to for events relating to the clipboard. The IClipboard object establishes this window as the system clipboard owner when you call IClipboard::empty (p. 162). If you call IClipboard::owner (p. 161) before calling empty, your window will not be returned because it is not yet the system clipboard owner.

You process clipboard events by creating and attaching an IClipboardHandler (p. 170) object to your clipboard owner window. In particular, if you use delayed rendering, you must attach an IClipboardHandler object to your clipboard's window (the owner window). The window dispatcher calls this handler when a request is made to the clipboard for data that has not been placed there yet.

Because the clipboard should only be kept open for a short time, create IClipboard objects as temporary objects with a short lifetime. This helps ensure that the clipboard is only open for the time necessary.

The IClipboard destructor always closes the clipboard if it is still open.

Public Functions

Clipboard Data Transfer

These members move data to and from the clipboard. You can request delayed rendering of data by calling setText, setBitmap, setHandle, or setData with a 0 data pointer or handle. For delayed rendering, create an IClipboard handler and attach it to the clipboard owner window to process requests to render the data when it is needed. Use delayed rendering when you have complex data formats or multiple data formats. It allows you to postpone moving the data to the clipboard until it is needed.

These members open the clipboard as needed and close it after transferring data, unless you have explicitly opened the clipboard by calling open (p. 162).

bitmap If data of the format IClipboard::bitmapFormat exists on the clipboard, this function creates a copy of the bitmap and returns its IBitmapHandle. An IInvalidRequest exception occurs if the format does not exist on the clipboard. You should call the function hasBitmap (p. 157) prior to calling this function to ensure that a bitmap exists on the clipboard.

```
virtual IBitmapHandle
  bitmap();
```

Win	PM	Motif
Y	Y	N

data Returns a void* value. This value can either be a pointer or handle to the data returned from the clipboard. It is your responsibility to know the type of the value because it is based on the format of the data.

This function always leaves the clipboard open. You must copy the data before closing the clipboard because you lose access to the data after you close the clipboard.

```
virtual void*
  data( const char* format );
```

Win	PM	Motif
Y	Y	N

format A valid clipboard format.

Exceptions	
IInvalidRequest	No data of the requested format exists on the clipboard.

hasBitmap If the clipboard has data with the format IClipboard::bitmapFormat, returns true.

```
virtual Boolean
  hasBitmap() const;
```

Win	PM	Motif
Y	Y	N

hasData If the clipboard has data of the requested format, returns true.

```
virtual Boolean
  hasData( const char* format ) const;
```

Win	PM	Motif
Y	Y	N

format A valid clipboard format.

hasText If the clipboard has data with the format IClipboard::textFormat, returns true.

```
virtual Boolean
  hasText() const;
```

Win	PM	Motif
Y	Y	N

setBitmap Copies the passed bitmap and places the handle on the clipboard with the format IClipboard::bitmapFormat.

This function opens the clipboard if it is not already open. It also closes it after placing the bitmap on the clipboard unless you have explicitly opened the clipboard by calling open (p. 162).

© IBM Corp. 1992, 1996

```
virtual IClipboard&
  setBitmap( const IBitmapHandle& bitmap );
```

Win	PM	Motif
Y	Y	N

bitmap The handle of a valid bitmap.

setData

Copies the passed data buffer and places it on the clipboard with the format specified. Register any private formats before placing data of the indicated format on the clipboard by calling registerFormat (p. 161). If the pointer to the data is 0, create an IClipboardHandler (p. 170) object to process requests to render the data.

This function opens the clipboard if it is not already open. It also closes it after placing the data on the clipboard unless you have explicitly opened the clipboard by calling open (p. 162).

```
virtual IClipboard&
  setData( const char* format,
           const void* data,
           unsigned long dataLength );
```

Win	PM	Motif
Y	Y	N

format A valid clipboard format.
data A pointer to a buffer containing the data of the indicated format.
dataLength
The length of the buffer stored in the *data* argument.

Exceptions	
IAccessError	The operating system's request to acquire shared storage failed.
IAccessError	The operating system's request to put the data on the clipboard failed.

setHandle

Makes *handle* shareable (so that it can be accessed by other processes) and places it on the clipboard. Use this function to copy bitmaps and metafiles to the clipboard. setBitmap (p. 157) uses this function to place the bitmap on the clipboard.

This function opens the clipboard if it is not already open. It also closes it after placing the handle on the clipboard unless you have explicitly opened the clipboard by calling open (p. 162).

```
virtual IClipboard&
  setHandle( const char* format,
             unsigned long handle );
```

Win	PM	Motif
Y	Y	N

format A valid clipboard format.
handle A handle to the object identified by the *format* parameter.

Exceptions	
IAccessError	The operating system's request to place the handle on the clipboard failed.

setText Copies *text* and places it on the clipboard with the format IClipboard::textFormat.

This function opens the clipboard if it is not already open. It also closes it after placing the text on the clipboard unless you have explicitly opened the clipboard by calling open (p. 162).

```
virtual IClipboard&
  setText( const char* text );
```

Win	PM	Motif
Y	*Y*	*N*

text A null-terminated character string.

text Returns IClipboard::textFormat data as an IString object.

If IClipboard::textFormat data exists on the clipboard, it returns the data as an IString object. An IInvalidRequest exception occurs if the format does not exist on the clipboard. Call hasText (p. 157) before calling this function to ensure that text exists on the clipboard.

This function opens the clipboard if it is not already open. It also closes it after retrieving the text from the clipboard unless you have explicitly opened the clipboard by calling open (p. 162).

```
virtual IString
  text();
```

Win	PM	Motif
Y	*Y*	*N*

Clipboard Formats

Use these members to register and query the clipboard formats. Register all private clipboard formats before using them.

format Returns the requested clipboard format. Use the returned format to query or set data on the clipboard.

1

```
static IString
  format( const Cursor& cursor );
```

Win	PM	Motif
Y	*Y*	*N*

Returns the format of the data at the cursor location.

cursor A valid clipboard cursor.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidParameter	The cursor is not valid.
IInvalidRequest	No data of the format indicated by the cursor exists on the clipboard.

2
```
static IString
  format( const FormatHandle& handle );
```

Win	PM	Motif
Y	Y	N

Returns the clipboard format of the format handle.

handle The handle of a valid clipboard format.

Exceptions	
IInvalidRequest	No data of the format handle exists on the clipboard. Ensure that the handle is valid.

formatAsHandle

Returns a clipboard format as a handle. Use this handle to perform clipboard operations that are not supported by the IClipboard class.

```
static FormatHandle
  formatAsHandle( const char* format );
```

Win	PM	Motif
Y	Y	N

format A valid clipboard format.

Exceptions	
IInvalidRequest	No data of the requested format exists on the clipboard.

formatCount

Returns the number of clipboard formats in the clipboard.

```
unsigned long
  formatCount() const;
```

Win	PM	Motif
Y	Y	N

primaryFormat

Returns the primary format of the data on the clipboard. The primary format is the first format you place on the clipboard after you call empty (p. 162). An IInvalidRequest results if there is no data on the clipboard.

```
static IString
  primaryFormat();
```

Win	PM	Motif
Y	Y	N

registerFormat

Registers *privateFormat* as a private format and returns its format handle.

```
static FormatHandle
  registerFormat( const char* privateFormat );
```

Win	PM	Motif
Y	Y	N

privateFormat
The text string that identifies your private clipboard format.

Exceptions	
IAccessError	The operating system's request to register the format failed. See the exception text provided with the exception for further details about the failure.

Clipboard Ownership

The *clipboard owner* is the window responsible for the data on the clipboard. IClipboard::empty (p. 162) establishes the window you provided on the IClipboard constructor as the clipboard owner. If you use delayed rendering, attach a handler to the clipboard owner window to process the delayed request to put the data on the clipboard.

owner

Returns the current clipboard owner window. The owner of the clipboard is established when the clipboard is emptied. If you have provided a clipboard window on the IClipboard constructor but have not yet emptied the clipboard, this function does not return your owner window unless your owner window was already the clipboard owner.

```
IWindowHandle
  owner() const;
```

Win	PM	Motif
Y	Y	N

Clipboard Setup

The clipboard setup members handle opening and closing the clipboard and clearing the initial contents of the clipboard.

close

Closes the clipboard.

If you manually open the clipboard by calling open (p. 162), you must also close the clipboard when you are finished transferring data. If you do not close the clipboard, other applications will wait indefinitely when they try to open the clipboard.

```
virtual IClipboard&
  close();
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The operating system's request to close the clipboard failed. See the text of the exception for further information.

empty

Clears the clipboard of all formats of data and establishes the window provided on the constructor as the clipboard owner.

This function opens the clipboard if it is not already open. It also closes it after use unless you have explicitly opened the clipboard by calling open (p. 162).

```
virtual IClipboard&
  empty();
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The operating system's request to empty the clipboard failed. See the text of the exception for further information.
IAccessError	The operating system request to establish the owner of the clipboard failed.

isOpen

Returns true if this IClipboard object has opened the clipboard.

```
virtual Boolean
  isOpen() const;
```

Win	PM	Motif
Y	Y	N

open

Opens the clipboard.

This function prevents other threads or processes from changing or examining the contents of the clipboard. Therefore, only keep the clipboard open for the time needed to place data on or remove data from the clipboard.

All IClipboard functions open the clipboard when needed. Unless you call this function to open the clipboard, they also close the clipboard when they are finished transferring data. Therefore, if you use this function to open the clipboard, you must also explicitly close the clipboard by calling close (p. 161).

```
virtual IClipboard&
  open();
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The operating system's request to open the clipboard failed. See the exception text for further information about the failure.

Constructors

Use these members to construct and destruct clipboard objects.

IClipboard

```
IClipboard( const IWindowHandle& clipboardWindow );
```

Win	PM	Motif
Y	Y	N

Creates clipboard objects. You can construct an IClipboard object by providing the window handle for the object to use for the owner of the clipboard.

clipboardWindow
A valid window handle used for clipboard data transfer. If you call empty (p. 162), it establishes this window as the owner of the clipboard.

Exceptions	
IInvalidParameter	The clipboard window handle is not valid.

~IClipboard

Calls close (p. 161) to close the clipboard if it is open.

```
virtual
  ~IClipboard();
```

Win	PM	Motif
Y	Y	N

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Clipboard Ownership

The *clipboard owner* is the window responsible for the data on the clipboard. IClipboard::empty (p. 162) establishes the window you provided on the IClipboard constructor as the clipboard

© IBM Corp. 1992, 1996

owner. If you use delayed rendering, attach a handler to the clipboard owner window to process the delayed request to put the data on the clipboard.

setOwner Establishes the owner of the clipboard. The owner of the clipboard controls the data on the clipboard and is responsible for responding to clipboard messages to render clipboard data. empty (p. 162) calls this function to establish the clipboard window you provide on the IClipboard constructor as the clipboard owner.

```
virtual IClipboard&
  setOwner();
```

Win	PM	Motif
N	Y	N

Exceptions	
IAccessError	The operating system's request to update the owner window failed. See the exception text for further details of the failure.

Public Data

Clipboard Predefined Formats

IClipboard provides a number of predefined clipboard formats that you can use for many standard data formats. You can use these formats without first registering them. You can also define additional clipboard formats by calling registerFormat (p. 161) before using them.

bitmapFormat

Defines a bitmap format for data on the clipboard.

```
static const char * const IC_IMPORTU * const
  bitmapFormat;
```

Win	PM	Motif
Y	Y	N

PM This format is implemented by the OS/2 format SZFMT_BITMAP.

Win This format is implemented by the Windows format CF_BITMAP.

displayBitmapFormat

Defines a bitmap representation of a private data format for clipboard data displayed in a Clipboard Viewer window. The data on the clipboard is identical to bitmapFormat (p. 164). Support this format if you do not support any of the standard clipboard formats.

```
static const char * const IC_IMPORTU * const
  displayBitmapFormat;
```

Win	PM	Motif
Y	Y	N

PM This format is implemented by the OS/2 format SZFMT_DSPBITMAP.

Win This format is implemented by the Windows format CF_DSPBITMAP.

displayMetafileFormat

Defines a metafile representation of a private data format displayed in a Clipboard Viewer window. The data on the clipboard is identical to metafileFormat (p. 165). Support this format if you do not support any of the standard clipboard formats.

```
static const char * const IC_IMPORTU * const
  displayMetafileFormat;
```

Win	PM	Motif
Y	*Y*	*N*

PM This format is implemented by the OS/2 format SZFMT_DSPMETAFILE.

Win This format is implemented by the Windows format CF_DSPENHMETAFILE.

displayTextFormat

Defines a text representation of a private data format displayed in a Clipboard Viewer window. The data on the clipboard is identical to textFormat (p. 166). Support this format if you do not support any of the standard clipboard formats.

```
static const char * const IC_IMPORTU * const
  displayTextFormat;
```

Win	PM	Motif
Y	*Y*	*N*

PM This format is implemented by the OS/2 format SZFMT_DSPTEXT.

Win This format is implemented by the Windows format CF_DSPTEXT.

metafileFormat

Defines a metafile format for data on the clipboard.

```
static const char * const IC_IMPORTU * const
  metafileFormat;
```

Win	PM	Motif
Y	*Y*	*N*

PM This format is implemented by the OS/2 format SZFMT_METAFILE.

Win This format is implemented by the Windows format CF_ENHMETAFILE.

paletteFormat

Defines a palette format for data on the clipboard.

```
static const char * const IC_IMPORTU * const
  paletteFormat;
```

Win	PM	Motif
Y	*Y*	*N*

PM This format is implemented by the OS/2 format SZFMT_PALETTE.

Win This format is implemented by the Windows format CF_PALETTE.

© IBM Corp. 1992, 1996

textFormat Defines a text format for data on the clipboard. The data is an array of text characters with a terminating null character.

```
static const char * const
  textFormat;
```

Win	PM	Motif
Y	*Y*	*N*

PM This format is implemented by the OS/2 format SZFMT_TEXT.

Win This format is implemented by the Windows CF_TEXT.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IClipboard contains the following nested classes:

IClipboard::Cursor (see page 167)

Nested Type Definitions

FormatHandle

```
typedef unsigned long FormatHandle;
```

A FormatHandle is a unique number (typically an atom) used by the operating system to identify a clipboard format.

IClipboard::Cursor

Derivation

IBase
  IVBase
    IClipboard::Cursor

Inherited By None.

Header File iclipbrd.hpp

Members

Member	Page	Member	Page
Constructor	167	setToFirst	168
Cursor	167	setToNext	168
invalidate	168	~Cursor	168
isValid	168		

The IClipboard::Cursor class is a nested cursor class used to iterate through the available formats of data in the clipboard. Unlike many other cursor classes, IClipboard::Cursor only supports movement in a forward direction. When you use a cursor loop like the one in the following example, the clipboard formats are returned in the same order that they were added.

```
// Iterate over clipboard formats...
IClipboard clipboard(mleHandle);
IClipboard::Cursor(clipboard);
for ( cursor.setToFirst(); cursor.isValid(); cursor.setToNext() )
  { // Do something with each format
    cout << "The format is: " << clipboard.format(cursor);
  }
```

Public Functions

Constructors

You can construct objects of this class from a reference to the window whose child windows are iterated. You can also destruct objects of this class.

Cursor You can construct objects of this class from a reference to an IClipboard object.

```
Cursor( IClipboard& clipboard );
```

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

IClipboard::Cursor

clipboard Reference to a clipboard object.

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	*Y*	*N*

Cursor Movement

You use the cursor movement members to position the clipboard cursor.

setToFirst Positions the cursor on the first format in the clipboard.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	*Y*	*N*

setToNext Positions the cursor on the next format in the clipboard. If you call setToNext before calling setToFirst, it positions the cursor on the first format in the clipboard.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	*Y*	*N*

Cursor Validation

Use the cursor validation members to determine if the clipboard cursor is valid or to indicate that it is no longer valid.

invalidate Puts the cursor in an invalid state.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	*Y*	*N*

isValid If the cursor is valid, returns true.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IClipboardHandler

Derivation

IBase
IVBase
IHandler
IClipboardHandler

Inherited By None.

Header File icliphdr.hpp

Members

Member	Page	Member	Page
Constructor	171	renderAllFormats	172
clipboardEmptied	172	renderFormat	172
dispatchHandlerEvent	171	~IClipboardHandler	171

The IClipboardHandler class processes the events that the clipboard sends to its owner. This includes requests to render clipboard data for formats that were put on the clipboard with delayed rendering.

The handler processes these events by creating an IEvent object and routing it to the appropriate virtual function. The virtual function allows you to supply your own specialized processing of the event. The return values from the virtual function specify whether the clipboard event is passed on to another handler object to be processed, as follows:

Value	Meaning
true	The IEvent has been handled and requires no additional processing.
false	The IEvent is passed to the next handler, as follows: • If there is another handler for the window, the event is passed on to the next handler. • If this is the last handler for the window, the event is passed on a call to the window's defaultProcedure function.

Public Functions

Constructors

Use these members to construct and destruct clipboard handler objects.

© Copyright IBM Corp. 1992, 1996

IClipboardHandler

Creates IClipboardHandler objects. Construct objects of this class only by using the default constructor, which does not take any arguments.

```
IClipboardHandler();
```

Win	PM	Motif
Y	*Y*	*N*

~IClipboardHandler

```
virtual
  ~IClipboardHandler();
```

Win	PM	Motif
Y	*Y*	*N*

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 the event to determine if it is appropriate for this handler object to process. If it is, the member calls the virtual function used to process the event.

dispatchHandlerEvent

Calls the appropriate virtual function to process clipboard events.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

© IBM Corp. 1992, 1996

event Reference to an event object.

Event Processing

These members are pure virtual functions that derived classes must implement. There is no default behavior for these functions.

clipboardEmptied

Called when the current clipboard owner needs to clean up data in the clipboard because another application window has requested ownership by clearing the data in the clipboard.

```
virtual Boolean
  clipboardEmptied( IEvent& event ) = 0;
```

Win	PM	Motif
Y	*Y*	*N*

event Reference to an event object.

renderAllFormats

Implemented by derived classes to render the data for all formats with delayed rendering. Place the data for all delayed formats into the clipboard with calls to setText, setBitmap, setMetafile, and setData.

Note: Do not open the clipboard during the processing of this function as you can hang the system. You can put data on the clipboard during the processing of this function without explicitly opening the clipboard.

```
virtual Boolean
  renderAllFormats( IEvent& event ) = 0;
```

Win	PM	Motif
Y	*Y*	*N*

event Reference to an event object.

renderFormat

Implemented by derived classes to render the data for a format with delayed rendering. Place data into the clipboard with setText, setBitmap, or setData.

Note: Do not open the clipboard during the processing of this function as you can hang the system. You can put data on the clipboard during the processing of this function without explicitly opening the clipboard.

```
virtual Boolean
  renderFormat( IEvent& event,
                const IString& format ) = 0;
```

Win	PM	Motif
Y	*Y*	*N*

event Reference to an event object.

format A valid clipboard format.

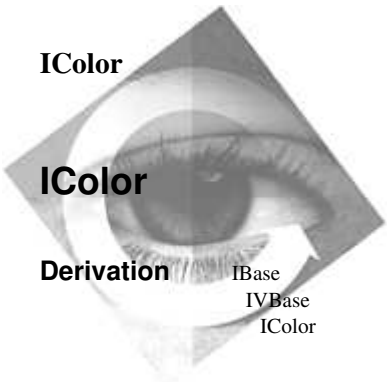
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IColor

Derivation

IBase
IVBase
IColor

Inherited By

IDeviceColor
IGUIColor

Header File

icolor.hpp

Members

Member	Page	Member	Page
Constructor	176	operator ==	176
asPixel	178	redMix	175
asRGBLong	178	setBlue	175
blueMix	174	setGreen	175
greenMix	175	setRed	175
index	178	systemColor	175
operator !=	176	value	175
operator =	177	~IColor	177

The IColor class allows you to query and set the color of various areas of windows represented by User Interface Class Library window objects. IColor is also the base class for other color classes.

Motif The color intensities of this class are 8-bit quantities and are represented as an unsigned char, resulting in intensity values from 0 to 255, with 255 the maximum intensity. X supports color intensities that are an unsigned short or 16 bits, and recognizes values from 0 to 65535.

Use the asPixel (p. 178) and the constructor that takes a pixel.

Public Functions

Color Mix

A color can be described by the amounts of the three primary colors (red, green, and blue) composing it. You can set and query any of these individual color components.

blueMix Returns the value of the blue component.

© Copyright IBM Corp. 1992, 1996

```
unsigned char
  blueMix() const;
```

Win	PM	Motif
Y	Y	Y

greenMix Returns the value of the green component.

```
unsigned char
  greenMix() const;
```

Win	PM	Motif
Y	Y	Y

redMix Returns the value of the red component.

```
unsigned char
  redMix() const;
```

Win	PM	Motif
Y	Y	Y

setBlue Sets the blue color mix.

```
IColor&
  setBlue( unsigned char blueMix );
```

Win	PM	Motif
Y	Y	Y

setGreen Sets the green color mix.

```
IColor&
  setGreen( unsigned char greenMix );
```

Win	PM	Motif
Y	Y	Y

setRed Sets the red color mix.

```
IColor&
  setRed( unsigned char redMix );
```

Win	PM	Motif
Y	Y	Y

systemColor Returns the SystemColor (p. 181) enumerator used to create this object.

```
SystemColor
  systemColor() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The object is a color specified other than by a SystemColor (p. 181) enumeration (for example, a color table index or red-green-blue mix instead).

value Returns the Color (p. 180) enumerator assigned to the object.

```
Color
  value() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Exceptions	
IInvalidRequest	The object is a color specified other than by a Color (p. 180) enumeration (for example, a color table index or red-green-blue mix instead).

Comparisons

Use these members to test whether two color objects represent the same color.

operator != Returns true if the colors are not identical.

```
Boolean
  operator !=( const IColor& color ) const;
```

Win	PM	Motif
Y	Y	Y

operator == Returns true if the colors are identical. The current red, green, and blue values are compared to determine identity.

```
Boolean
  operator ==( const IColor& color ) const;
```

Win	PM	Motif
Y	Y	Y

Constructors

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

IColor

1

```
IColor( Color color );
```

Win	PM	Motif
Y	Y	Y

color A Color (p. 180) enumerator value identifying a specific color.

2

```
IColor( unsigned char red,
        unsigned char green,
        unsigned char blue );
```

Win	PM	Motif
Y	Y	Y

The color is identified by the combination of the specified red, green, and blue values.

Use this constructor to identify a color not defined by the Color (p. 180) enumeration.

3

```
IColor( const IColor& color );
```

Win	PM	Motif
Y	Y	Y

The color is copied from the specified object.

4 `IColor( long index );`

Win	PM	Motif
Y	Y	Y

index The index of the color to use.

PM *index* represents an entry in the color table associated with the desktop window.

Win *index* represents an entry in the system default palette.

Exceptions	
IInvalidRequest	*index* has an invalid value.

5 `IColor( unsigned long pixel );`

Win	PM	Motif
N	N	Y

pixel A pixel index from the default color map of the current display.

6 `IColor( SystemColor value );`

Win	PM	Motif
Y	Y	Y

Use this constructor to create an IColor object whose color is linked to the system color setting for a screen area. The color returned by the object will represent the current system setting for the screen area, even if the system settings are changed after the IColor object is created. When you use IColor objects created with this constructor to set the color of an IWindow object, the window is updated automatically with the new color when the system settings are changed.

If you use the setBlue (p. 175), setRed (p. 175), or setGreen (p. 175) functions to modify the IColor object, the link to the system color setting is broken.

value A SystemColor (p. 181) enumerator value identifying a screen area.

operator =

Replaces the color. If the IColor object on the right hand side of the = was created using the constructor that accepts a SystemColor (p. 181) value and still is linked to the system color area, the link is copied also.

```
IColor&
  operator =( const IColor& color );
```

Win	PM	Motif
Y	Y	Y

~IColor

© IBM Corp. 1992, 1996

IColor

```
virtual
  ~IColor();
```

Win	PM	Motif
Y	*Y*	*Y*

Conversions

Use these members to convert a color object into a value that can be used with APIs provided by the windowing system.

asPixel Returns the default color-map index that most closely matches the color's combination of red, green, and blue values.

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

Win	PM	Motif
N	*N*	*Y*

asRGBLong Returns the red, green, and blue color values combined into a long integer.

```
long
  asRGBLong() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif You can use this function to obtain the color as a long value and; then convert it to the hexadecimal format #RRGGBB. X11 recognizes this format; however, this format has been discouraged as of Release 5 of XLib (X11R5).

index Returns the logical color table index closest to the red, green, and blue values.

```
virtual long
  index() const;
```

Win	PM	Motif
Y	*Y*	*Y*

PM The following list contains the index values and colors found in a logical color table:

Index	Color
-2	White
-1	Black
0	Background color of the device (white)
1	Blue
2	Red
3	Pink
4	Green
5	Cyan
6	Yellow
7	Neutral (black)
8	Dark gray
9	Dark blue
10	Dark red
11	Dark pink

12	Dark green
13	Dark cyan
14	Brown
15	Pale gray

Win The following list contains the index values and colors found in the default system palette:

Index	Color
0	Black
1	Dark red
2	Dark green
3	Brown
4	Dark blue
5	Dark pink
6	Dark cyan
7	Pale gray
12	Dark gray
13	Red
14	Green
15	Yellow
16	Blue
17	Pink
18	Cyan
19	White

Motif The following list contains the index values and colors found in a logical color table. The values in parentheses show the corresponding color from the file rgb.txt.

Index	Color
0	White (white)
1	Blue (blue)
2	Red (red)
3	Pink (pink)
4	Green (green)
5	Cyan (cyan)
6	Yellow (yellow)
7	Black (black)
8	Dark gray (dark slate gray)
9	Dark blue (dark slate blue)
10	Dark red (red4)
11	Dark pink (deep pink)
12	Dark green (dark green)
13	Dark cyan (cyan4)
14	Brown (brown)

© IBM Corp. 1992, 1996

15 Pale gray (light gray)

Exceptions	
IInvalidRequest	The object cannot be represented by a color table index.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

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

IColor

```
IColor();
```

Win	PM	Motif
Y	*Y*	*Y*

This constructor sets the red, green, and blue color components to 0. If not changed by the derived class, the resulting color object represents black.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Color

```
Color {
    white,     blue,      red,       pink,      green,
    cyan,      yellow,    black,     darkGray,  darkBlue,
    darkRed,   darkPink,  darkGreen, darkCyan,  brown,
    paleGray
    };
```

Use these enumerator values to identify a specific color.

SystemColor

```
SystemColor {
    shadowIconHiliteBgnd,  shadowIconHiliteFgnd,  shadowIconText,
    entryFieldBgnd,        listBoxBgnd,           disableMenuText,
    menuHiliteText,        menuHiliteBgnd,        notebookPageBgnd,
    inactiveScrollBar,     defaultControl,        buttonLight,
    buttonMiddle,          buttonDark,            defaultButton,
    titleLine,             menuShadow,            dialogShadow,
    iconText,              dialogBgnd,            hiliteFgnd,
    hiliteBgnd,            inactiveTitleTextBgnd, activeTitleTextBgnd,
    inactiveTitleText,     activeTitleText,       outputText,
    windowStaticText,      scrollBar,             desktopBgnd,
    activeTitleBgnd,       inactiveTitleBgnd,     menuBgnd,
    windowBgnd,            frameBorder,           menuText,
    windowText,            titleText,             sizeBar,
    scrollArrow,           activeFrameBorder,     inactiveFrameBorder,
    mainWindowBgnd,        helpWindowBgnd,        helpText,
    helpHiliteText
    };
```

Use these enumerator values to identify a specific system color. System colors are the colors assigned to specific screen areas.

PM Each SystemColor value corresponds to one of the system SYSCLR_* colors in Presentation Manager.

Win Each SystemColor value corresponds to one of the system COLOR_* colors in the Windows SDK. The only exception to this is defaultButton, whose value is 0, 0, 0 (black).

© IBM Corp. 1992, 1996

IComboBox

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IEntryField
IBaseComboBox
IComboBox

Inherited By None.

Header File icombobx.hpp

Members

Member	Page	Member	Page
Constructor	188	defaultStyle	190
add	183	remove	189
addAscending	184	removeAll	189
addAsFirst	185	removeAt	189
addAsLast	186	removeId	193
addAsNext	187	replaceAt	189
addDescending	187	setDefaultStyle	190
addId	193	~IComboBox	189
classDefaultStyle	193		

The IComboBox class extends the IBaseComboBox combination box's control window creation and management to include adding, removing and replacing items in the combination box list box.

Handlers derived from the following classes handle events for IComboBox objects:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISelectHandler (p. 851)
- IShowListHandler (p. 870)

Motif Handlers derived from IEditVerifyHandler can be attached to IComboBox objects.

© Copyright IBM Corp. 1992, 1996

Public Functions

Add Items

Use these members to add items to the list box portion of the combination box. You can add items to the list box at the following positions:

- A specified location given by an index or cursor
- The first item
- The last item
- The next item following a cursor
- In descending sort order
- In ascending sort order

Represent the new text item by either a resource identifier or the text string itself.

add Inserts the line of text at a specified location in the list box portion of the combination box.

1

```
virtual IComboBox&
  add( const char* text,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box portion of the combination box at the cursor position and sets the cursor to the inserted item.

text The text you want to insert.
cursor A combination box cursor object. The cursor's position identifies where to insert the item in the list.

2

```
virtual unsigned long
  add( unsigned long index,
       const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box portion of the combination box and returns the index of the item.

text The text you want to insert.
index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

3

```
virtual unsigned long
  add( unsigned long index,
       const char * const* itemList,
       unsigned long count = 1 );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Inserts items corresponding to the itemList text strings into the list box portion of the combination box and returns the index of the first item inserted.

index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

itemList An array of character strings.

count Number of text items in itemList. The default is 1 item.

4
```
virtual unsigned long
  add( unsigned long index,
       const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts text corresponding to the resource item into the list box portion of the combination box and returns the index of the item.

index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

item Resource ID of the text you want to insert in the list.

5
```
virtual IComboBox&
  add( const IResourceId& item,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box portion of the combination box at the cursor position and sets the cursor to the inserted item.

item Resource ID of the text you want to insert in the list.
cursor A combination box cursor object. The cursor's position identifies where to insert the item in the list.

addAscending

Inserts the line of text in ascending sort order.

1
```
virtual unsigned long
  addAscending( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box portion of the combination box in ascending sort order and returns the index of the inserted item.

text The text you want to insert.

2
```
virtual unsigned long
  addAscending( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, into the list box portion of the combination box in ascending sort order and returns the index of the inserted item.

item Resource ID of the text you want to insert in the list.

addAsFirst

Inserts the line of text as the first item in the list box portion of the combination box.

1
```
virtual unsigned long
  addAsFirst( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the first item in the list box portion of the combination box.

text The text you want to insert.

2
```
virtual unsigned long
  addAsFirst( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the first item in the list box portion of the combination box.

item Resource ID of the text you want to insert in the list.

3
```
virtual IComboBox&
  addAsFirst( const char* text,
              Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the first item in the list box portion of the combination box and puts the cursor on the inserted item.

text The text you want to insert.
cursor A combination box cursor object. The cursor is set to the first item in the list.

4
```
virtual IComboBox&
  addAsFirst( const IResourceId& item,
              Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box portion of the combination box and puts the cursor on that item.

© IBM Corp. 1992, 1996

item Resource ID of the text you want to insert in the list.
cursor A combination box cursor object. The cursor is set to the first item in the list.

addAsLast Inserts the line of text as the last item in the list box portion of the combination box.

1
```
virtual unsigned long
  addAsLast( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box portion of the combination box.

item Resource ID of the text you want to insert in the list.

2
```
virtual unsigned long
  addAsLast( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the last item in the list box portion of the combination box.

text The text you want to insert.

3
```
virtual IComboBox&
  addAsLast( const char* text,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the last item in the list box portion of the combination box and puts the cursor on that item.

text The text you want to insert.
cursor A combination box cursor object. The cursor is set to the last item in the list.

4
```
virtual IComboBox&
  addAsLast( const IResourceId& item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box portion of the combination box and puts the cursor on that item.

item Resource ID of the text you want to insert in the list.
cursor A combination box cursor object. The cursor is set to the last item in the list.

addAsNext

Inserts the line of text in the list box portion of the combination box after the cursor.

1

```
virtual IComboBox&
  addAsNext( const char* text,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text after the current cursor position in the list box portion of the combination box and places the cursor on that item.

text The text you want to insert.
cursor A combination box cursor object. The cursor's position identifies where to insert the item in the list.

2

```
virtual IComboBox&
  addAsNext( const IResourceId& item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts a new text item, corresponding to the resource identifier, after the current cursor position in the list box portion of the combination box and places the cursor on that item.

item Resource ID of the text you want to insert in the list.
cursor A combination box cursor object. The cursor's position identifies where to insert the item in the list.

addDescending

Inserts the line of text in descending sort order.

1

```
virtual unsigned long
  addDescending( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box portion of the combination box in descending sort order and returns the index of the inserted item.

text The text you want to insert.

2

```
virtual unsigned long
  addDescending( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, into the list box portion of the combination box in descending sort order and returns the index of the inserted item.

© IBM Corp. 1992, 1996

item Resource ID of the text you want to insert in the list.

Constructors

You can construct and destruct objects of this class.

IComboBox

1
```
IComboBox( unsigned long id,
           IWindow* parent );
```

Win	PM	Motif
Y	Y	N

You can construct objects of this class using the parent window and a combination box ID.

id A combination box control ID.
parent The parent window.

2
```
IComboBox( unsigned long id,
           IWindow* parent,
           IWindow* owner,
           const IRectangle& initial = IRectangle ( ),
           const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class using the ID, parent, owner, size, position, and style parameters.

id A combination box control ID.

parent The parent window.

owner The owner window.

initial The initial position and size of the combination box you construct. The default is the rectangle constructed by the default IRectangle constructor. Optional.

style The combination box's characteristics. Optional.

3
```
IComboBox( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	N

You can construct objects of this class using the handle of an existing combination box window.

handle The window handle of an existing combination box control.

~IComboBox

```
virtual
  ~IComboBox();
```

Win	PM	Motif
Y	Y	Y

Remove and Replace Items

Use these members to remove items from or replace items in the list box portion of the combination box.

remove Removes the specified item from the list box and returns the number of items that remain.

index The index of the text item to remove.

```
virtual unsigned long
  remove( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

removeAll Removes all items from the list box.

```
virtual IComboBox&
  removeAll();
```

Win	PM	Motif
Y	Y	Y

removeAt Removes the item at the cursor position and sets the cursor to the item following the removed item.

```
virtual IComboBox&
  removeAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

replaceAt Replaces the list box item at the cursor position. An invalid cursor causes an exception.

1

```
virtual IComboBox&
  replaceAt( const char* text,
             const Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Replaces the item's string at the cursor with the new text.

text The text string.
cursor A combination box cursor object. It must be valid.

2

```
virtual IComboBox&
  replaceAt( const IResourceId& item,
             const Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Replaces the item's string at the cursor with the text corresponding to the resource identifier.

item The resource ID of the text string to replace in the list box portion of the combination box.

cursor A combination box cursor object. It must be valid.

Styles

Use these members to define, set, and retrieve the IComboBox default style.

Note: The IComboBox class inherits its style behavior from IBaseComboBox except for the default style members described here.

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 193) unless you have changed it using setDefaultStyle (p. 190).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent combination boxes.

style Use the styles provided by IComboBox Styles (p. 192) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IBaseComboBox		
convertToGUIStyle	itemHandle	**setDefaultStyle**
count	itemText	setForegroundColor
defaultStyle	layoutAdjustment	setItemHandle
deselect	limit	setItemText
deselectAll	locateText	setLimit
elementAt	minimumRows	setMinimumRows
enableNotification	moveSizeTo	setTop
hasFocus	nativeRect	showList

IComboBox

IBaseComboBox		
hideList	numberOfSelections	size
isEmpty	position	top
isHorizontalScroll	select	topHandle
isListShowing	selection	type
isSelected	setBackgroundColor	visibleRectangle

IEntryField		
alignment	enable	isWriteable
backgroundColor	enableAutoScroll	leftIndex
charType	enableAutoTab	limit
clear	enableCommand	moveSizeTo
convertToGUIStyle	enableDataUpdate	paste
copy	enableInsertMode	removeAll
cursorPosition	enableMargin	resetTextChangedFlag
cut	enableNotification	selectedRange
defaultStyle	foregroundColor	selectedText
disable	hasSelectedText	selectedTextLength
disableAutoScroll	hasTextChanged	selectRange
disableAutoTab	isAutoScroll	setAlignment
disableCommand	isAutoTab	setCharType
disableDataUpdate	isCommand	setCursorPosition
disableInsertMode	isEmpty	**setDefaultStyle**
disableMargin	isInsertMode	setLeftIndex
discard	isMargin	setLimit

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

IComboBox

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

IBaseComboBox		
calcMinimumSize	incrementChangeCount	setLayoutDistorted
changeCount	registerCallbacks	unregisterCallbacks

IEntryField		
calcMinimumSize	isDragStarting	registerCallbacks
initialize	passEventToOwner	setLayoutDistorted

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

These INotificationId members define the possible notifications that IComboBox provides to its observers. The following events can occur:

- Add a list box item
- Remove a list box item

addId This notification identifier is the notification IComboBox objects provide their observers when an item is added to the list box portion of the combination box. The number of items added is provided in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  addId;
```

Win	PM	Motif
Y	Y	Y

removeId This notification identifier is the notification IComboBox objects provide their observers when an item is removed to the list box portion of the combination box. The number of items removed is provided in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). If the value is 0, all items were removed.

```
static INotificationId const
  removeId;
```

Win	PM	Motif
Y	Y	Y

Styles

Use these members to define, set, and retrieve the IComboBox default style.

Note: The IComboBox class inherits its style behavior from IBaseComboBox except for the default style members described here.

classDefaultStyle

Provides the original default style for this class, which is the following: IComboBox::simpleType | IComboBox::anyData | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

IBaseComboBox		
anyData	dropDownType	notFound
autoScroll	enterId	oemData
border3D	first	readOnlyDropDownType
classDefaultStyle	horizontalScroll	sbcsData
dbcsData	mixedData	selectId

© IBM Corp. 1992, 1996

IComboBox

IEntryField		
characterTypeId	dataUpdateId	end

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IComboBox contains the following nested classes:

IComboBox::Style (see page 196)

Nested Type Definitions

Cursor `typedef IBaseComboBox::Cursor Cursor;`

This typedef supports prior use of the IComboBox::Cursor class, which now exists in IBaseComboBox.

© IBM Corp. 1992, 1996

IComboBox::Style

Derivation

IBase
IBitFlag
IComboBox::Style

Inherited By None.

Header File icombobx.hpp

The nested class IComboBox::Style provides a set of valid styles for the IComboBox (p. 182) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IComboBoxNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          IEntryFieldNotifyHandler
            IComboBoxNotifyHandler

Inherited By None.

Header File icombonh.hpp

Members

Member	Page
Constructor	197
dispatchHandlerEvent	198
~IComboBoxNotifyHandler	197

The IComboBoxNotifyHandler class processes events for all classes of combo boxes.

This class is designed to handle events that require the combo box class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IComboBoxNotifyHandler

This is the default constructor and accepts no parameters.

```
IComboBoxNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IComboBoxNotifyHandler

© Copyright IBM Corp. 1992, 1996

IComboBoxNotifyHandler

```
virtual
  ~IComboBoxNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the combo box observers if the following event is received:

- selected event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IEntryFieldNotifyHandler		
dispatchHandlerEvent		

ITextControlNotifyHandler		
dispatchHandlerEvent		

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ICommand

Derivation IBase
ICommand

Inherited By None.

Header File icmd.hpp

Members

Member	Page	Member	Page
kBoldId	201	kLocateId	202
kCancelId	201	kMaximizeId	204
kCloseId	204	kMinimizeId	204
kCopyId	201	kMoveId	204
kCopyToId	201	kOkId	203
kCutId	201	kOpenId	203
kFileApplyId	201	kPasteId	203
kFileCancelId	201	kPrintId	203
kFileOkId	202	kRestoreId	204
kFontApplyId	202	kSaveId	203
kFontCancelId	202	kSettingsId	203
kFontOkId	202	kSizeId	204
kHelpId	202	kUnderscoreId	203
kHideId	204	kWindowListId	205
kItalicId	202		

The ICommand class contains common command-related elements that can be used in more than one class. Commands are events that can be generated from push buttons, menu items, or accelerator keys, and processed by the ICommandHandler (p. 214) class.

You cannot create objects of this class, because ICommand has no public constructors.

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© Copyright IBM Corp. 1992, 1996

Public Data

Common Application Commands

ICommand provides a set of CommandId values that you can use as the identifiers for common application commands.

kBoldId The application command generated by a **bold** tool bar button. This value is equivalent to the IC_ID_BOLD define in the file icconst.h.

```
static const CommandId
  kBoldId;
```

Win	PM	Motif
Y	*Y*	*N*

kCancelId The application command generated by pressing the Esc key on a dialog.

```
static const CommandId
  kCancelId;
```

Win	PM	Motif
Y	*Y*	*N*

kCopyId The application command generated by a **copy** tool bar button. This value is equivalent to the IC_ID_COPY define in the file icconst.h.

```
static const CommandId
  kCopyId;
```

Win	PM	Motif
Y	*Y*	*N*

kCopyToId The application command generated by a **copy to** tool bar button. This value is equivalent to the IC_ID_COPYTO define in the file icconst.h.

```
static const CommandId
  kCopyToId;
```

Win	PM	Motif
Y	*Y*	*N*

kCutId The application command generated by a **cut** tool bar button. This value is equivalent to the IC_ID_CUT define in the file icconst.h.

```
static const CommandId
  kCutId;
```

Win	PM	Motif
Y	*Y*	*N*

kFileApplyId The application command generated by the Apply push button on a file dialog.

```
static const CommandId
  kFileApplyId;
```

Win	PM	Motif
Y	*Y*	*N*

kFileCancelId The application command generated by the Cancel push button on a file dialog.

© IBM Corp. 1992, 1996

```
static const CommandId
  kFileCancelId;
```

Win	PM	Motif
Y	*Y*	*N*

kFileOkId The application command generated by the OK push button on a file dialog.

```
static const CommandId
  kFileOkId;
```

Win	PM	Motif
Y	*Y*	*N*

kFontApplyId The application command generated by the Apply push button on a font dialog.

```
static const CommandId
  kFontApplyId;
```

Win	PM	Motif
Y	*Y*	*N*

kFontCancelId

The application command generated by the Cancel push button on a font dialog.

```
static const CommandId
  kFontCancelId;
```

Win	PM	Motif
Y	*Y*	*N*

kFontOkId The application command generated by the OK push button on a font dialog.

```
static const CommandId
  kFontOkId;
```

Win	PM	Motif
Y	*Y*	*N*

kHelpId The application command generated by a **help** tool bar button. This value is equivalent to the IC_ID_HELP define in the file icconst.h.

```
static const CommandId
  kHelpId;
```

Win	PM	Motif
Y	*Y*	*N*

kItalicId The application command generated by an **italic** tool bar button. This value is equivalent to the IC_ID_ITALIC define in the file icconst.h.

```
static const CommandId
  kItalicId;
```

Win	PM	Motif
Y	*Y*	*N*

kLocateId The application command generated by a **locate** tool bar button. This value is equivalent to the IC_ID_LOCATE define in the file icconst.h.

```
static const CommandId
  kLocateId;
```

Win	PM	Motif
Y	*Y*	*N*

kOkId The application command that generally results from selecting an OK push button, or pressing the Enter key on a dialog without a default push button.

```
static const CommandId
  kOkId;
```

Win	PM	Motif
Y	*Y*	*N*

kOpenId The application command generated by a **open** tool bar button. This value is equivalent to the IC_ID_OPEN define in the file icconst.h.

```
static const CommandId
  kOpenId;
```

Win	PM	Motif
Y	*Y*	*N*

kPasteId The application command generated by a **paste** tool bar button. This value is equivalent to the IC_ID_PASTE define in the file icconst.h.

```
static const CommandId
  kPasteId;
```

Win	PM	Motif
Y	*Y*	*N*

kPrintId The application command generated by a **print** tool bar button. This value is equivalent to the IC_ID_PRINT define in the file icconst.h.

```
static const CommandId
  kPrintId;
```

Win	PM	Motif
Y	*Y*	*N*

kSaveId The application command generated by a **save** tool bar button. This value is equivalent to the IC_ID_SAVE define in the file icconst.h.

```
static const CommandId
  kSaveId;
```

Win	PM	Motif
Y	*Y*	*N*

kSettingsId The application command generated by a **setting** tool bar button. This value is equivalent to the IC_ID_SETTINGS define in the file icconst.h.

```
static const CommandId
  kSettingsId;
```

Win	PM	Motif
Y	*Y*	*N*

kUnderscoreId

The application command generated by a **underscore** tool bar button. This value is equivalent to the IC_ID_UNDERSCORE define in the file icconst.h.

```
static const CommandId
  kUnderscoreId;
```

Win	PM	Motif
Y	*Y*	*N*

© IBM Corp. 1992, 1996

Common System Commands

ICommand provides a set of CommandId values that you can use as the identifiers for portable system commands.

kCloseId The system command that closes a frame window.

```
static const CommandId
  kCloseId;
```

Win	PM	Motif
Y	Y	N

kHideId The system command that hides a frame window.

```
static const CommandId
  kHideId;
```

Win	PM	Motif
Y	Y	N

Win This command is equivalent to kMinimizeId (p. 204).

kMaximizeId The system command that maximizes the size of a frame window.

```
static const CommandId
  kMaximizeId;
```

Win	PM	Motif
Y	Y	N

kMinimizeId The system command that minimizes the size of a frame window.

```
static const CommandId
  kMinimizeId;
```

Win	PM	Motif
Y	Y	N

kMoveId The system command the allows the user to move a frame window if it has a title bar.

```
static const CommandId
  kMoveId;
```

Win	PM	Motif
Y	Y	N

kRestoreId The system command that restores a frame window to the size and position it had before last being minimized, maximized, or hidden.

```
static const CommandId
  kRestoreId;
```

Win	PM	Motif
Y	Y	N

kSizeId The system command that allows the user to size a frame window if it has a sizing border.

```
static const CommandId
  kSizeId;
```

Win	PM	Motif
Y	Y	N

kWindowListId

The system command that causes the presentation system to show the list of windows that the user can switch to.

```
static const CommandId
  kWindowListId;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

ActionType

```
ActionType {
    applicationCommand,
    systemCommand,
    help
    };
```

The following enumerators identify the type of action run by an accelerator key. You use these enumerators when constructing an IAcceleratorKey object (p. 25) and calling the function IAcceleratorKey::actionType (p. 23).

applicationCommand
The accelerator key runs an application command.

systemCommand
The accelerator key runs a system command.

help
The accelerator key issues a help request.

CommandId

```
typedef unsigned long CommandId;
```

This typedef helps document that the CommandId values provided by this class can be used as identifiers for commands and system commands in the IAcceleratorKey (p. 22) class. The use of this typedef also allows you to use command identifiers not provided by ICommand with IAcceleratorKey.

© IBM Corp. 1992, 1996

ICommandConnectionTo

Derivation

IBase
IVBase
IHandler
ICommandHandler
ICommandConnectionTo

Inherited By None.

Header File icmdhdr.hpp

Members

Member	Page	Member	Page
Constructor	207	systemCommand	208
command	208	~ICommandConnectionTo	207

The ICommandConnectionTo class is a template class, derived from ICommandHandler (p. 214), that processes application and system command events and routes them to the template argument class. This class allows you to process command events in objects that do not derive from ICommandHandler without having to manually derive from ICommandHandler, override the command (p. 217) and systemCommand (p. 217) functions, and pass the events to the object.

The ICommandConnectionTo class routes both command and system command events to the same member of the template argument class. You can use the eventId member of ICommandEvent to distinguish between these two events.

To use the ICommandConnectionTo class, follow these steps:

1. Instantiate the ICommandConnectionTo template with a class containing a member function whose signature and behavior are the same as ICommandHandler::command (p. 217).
2. Construct an object of the new template class by passing the object and the address of the member function handling the command events.
3. Attach the template command handler by using IHandler::handleEventsFor (p. 413) to pass the appropriate window to the command handler.

You use an ICommandConnectionTo object anywhere you use an ICommandHandler object. See the class description of ICommandHandler for a description of the uses and limitations of these classes.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

You can construct and destruct template objects of this class. You cannot copy or assign objects of this class.

ICommandConnectionTo

Constructs the ICommandConnectionTo object.

```
ICommandConnectionTo( ATarget& target,
                      MemberFunction memberFunction );
```

Win	PM	Motif
Y	*N*	*N*

The parameters are the following:

target	The object to receive command events.
memberFunction	The address of the member function to receive command events.

~ICommandConnectionTo

```
virtual
  ~ICommandConnectionTo();
```

Win	PM	Motif
Y	*N*	*N*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

Protected Functions

Event Processing

These members are overridden to route command and system command events to the target object provided on construction.

command Overridden to route command events to the member function of the target object provided on construction.

```
virtual Boolean
  command( ICommandEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

systemCommand

Overridden to route system command events to the member function of the target object provided on construction.

```
virtual Boolean
  systemCommand( ICommandEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

Inherited Protected Functions

ICommandHandler		
command	dispatchHandlerEvent	systemCommand

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

MemberFunction

```
typedef Boolean (ATarget::*MemberFunction)(ICommandEvent&);
```

A pointer to a member function of Class ATarget that takes a reference to an ICommandEvent and returns a Boolean.

© IBM Corp. 1992, 1996

ICommandEvent

Derivation

IBase
IVBase
IEvent
ICommandEvent

Inherited By None.

Header File icmdevt.hpp

Members

Member	Page	Member	Page
Constructor	212	source	211
commandId	211	~ICommandEvent	212
isFromFrame	211		

The ICommandEvent class represents an application-specific or system command. ICommandHandler (p. 214) constructs and processes objects of this class, typically as a result of the user selecting one of the following:

- A menu item
- A push button
- An accelerator key sequence

If these preceding controls each use the same value for the following identifiers, then all three produce the same command ID when pressed or selected:

- Window ID for a button
- Menu item ID for a menu item
- Accelerator command value for the accelerator key

Subsequently, the command ID is passed in the ICommandEvent. This allows an application to unify the logic for all three types of input within a single ICommandHandler.

PM In Presentation Manager, the event result for WM_COMMAND and WM_SYSCOMMAND is reserved. Therefore, your callback functions do not have to set any result before they return true.

Presentation Manager's default window procedure does not pass an unprocessed WM_COMMAND or WM_SYSCOMMAND message to another window, either up the owner chain or up the parent window chain.

© Copyright IBM Corp. 1992, 1996

In OS/2, ICommandEvents from accelerators go directly to the associated IFrameWindow. For portability, attach the ICommandHandler to the IFrameWindow, which has the following characteristics:

- Has these accelerators
- Is the owner of this menu
- Is the owner of these push buttons, either directly or by owning canvases, which, in turn, own the push buttons

Public Functions

Command Information

A command event contains the ID of the command to run and limited information about the source of the command.

commandId Returns the ID of the push button or menu item, or the command ID of the accelerator key that initiated the command event. Application code can also create and post command events. In this case, the returned ID is whatever the creator of the event specified.

```
unsigned long
  commandId() const;
```

Win	PM	Motif
Y	Y	Y

isFromFrame Returns a Boolean type value (true or false) indicating whether the event has already been dispatched to the frame window and, consequently, whether it should not be dispatched to the frame again.

```
Boolean
  isFromFrame() const;
```

Win	PM	Motif
Y	Y	Y

source Returns the enumerator corresponding to the way in which the command event was initiated. The enumeration Source (p. 213) provides the return values. It is possible for an application to have a menu item, a push button, and an accelerator table entry that all generate the same command ID. You can use this function to determine which of the three was the source of the command event.

```
Source
  source() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

© IBM Corp. 1992, 1996

ICommandEvent

ICommandEvent

Construct an ICommandEvent from the specified event. ICommandHandler::dispatchHandlerEvent (p. 217) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function ICommandHandler::command (p. 217) or ICommandHandler::systemCommand (p. 217).

```
ICommandEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~ICommandEvent

```
virtual
  ~ICommandEvent();
```

Win	PM	Motif
Y	Y	Y

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**	

Source

```
Source {
    pushButton,
    menu,
    accelerator,
    unknown
    };
```

These enumerators list the possible origin of the command event that ICommandEvent::source (p. 211) can return:

pushButton
The origin of the command event is a push button.

menu
The origin of the command event is a menu.

accelerator
The origin of the command event is an accelerator.

unknown
The origin of the command event is unknown.

© IBM Corp. 1992, 1996

ICommandHandler

Derivation

IBase
IVBase
IHandler
ICommandHandler

Inherited By

ICommandConnectionTo
IMMPlayerPanelHandler

Header File

icmdhdr.hpp

Members

Member	Page	Member	Page
Constructor	216	systemCommand	217
command	217	~ICommandHandler	216
dispatchHandlerEvent	217		

The ICommandHandler class processes application command and system command events.

Create a handler derived from ICommandHandler and attach it to the following, as needed:

- The client window
- The frame window
- Another appropriate window, such as the owner window of a push button

You can attach a command handler by using IHandler::handleEventsFor (p. 413) to pass the appropriate window to the command handler.

When the command handler receives a command event, it creates an ICommandEvent (p. 210) object and routes that object to the appropriate ICommandHandler virtual function. You must override these virtual functions to supply your own specialized processing of a command event.

Push buttons and menu items (including pop-up menu items) send a command event to their owner window. Accelerators send a command event to their frame window. Only frames, canvases, and containers have support to pass on an unprocessed command event. Canvases and containers forward events to their owner window. A frame window passes application commands and the SC_CLOSE system command to its client window.

© Copyright IBM Corp. 1992, 1996

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 Pass on the command event for additional processing, as follows:

- If there is another handler for the window, pass the command event to the next handler.
- If this is the last handler for the window, call IWindow::defaultProcedure (p. 1082) to process the command event.

You can attach an ICommandHandler directly to a control, but the control must be its own owner in this case. However, an owner window is also used for passing on unprocessed keyboard and mouse events. As a result, if you choose to have a control own itself, you lose substantial support for that control. For example, if you have a push button own itself, you lose all dialog-box support for the push button, such as Tab-key support and default push button support.

You can attach the following:

- The same handler to more than one window
- More than one handler, of either the same or different types, to the same window

Note: Because IFrameHandler (p. 342) and ICnrMenuHandler (Vol. III) also processes ICommandEvents, think of them as if you had more than one ICommandHandler attached to a window. If you attach an IFrameHandler or ICnrMenuHandler to the same window as an ICommandHandler, your ICommandHandler might not be called, depending on the order in which you attached these handlers.

PM If the frame window is a page window of a notebook, the notebook does not use the accelerators of that frame. Only the accelerators of the frame window above the notebook in the parent chain are used.

Motif Motif has no concept of command events. However, an event that is a logical abstraction above button press, menu selection, and accelerator-key press is a useful construct. Therefore, the User Interface Class Library provides a Motif implementation that supports ICommandEvents, their creation, propagation, and handling.

These events are implemented as true X events. The library uses an unmaskable client message event with event data that not only identifies it as a command event, but also provides all other information needed to construct an ICommandEvent. The control classes derived from IWindow, which in Presentation Manager produce WM_COMMAND messages, are responsible for generating this X client message

© IBM Corp. 1992, 1996

event. The User Interface Class Library uses IWindowHandle::sendEvent (p. 1109) to send the event to X. The control sends the event to its owner widget, which matches Presentation Manager's system behavior for routing WM_COMMAND messages.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICommandHandler

Constructs the default command handler.

```
ICommandHandler();
```

Win	PM	Motif
Y	Y	Y

~ICommandHandler

```
virtual
  ~ICommandHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An ICommandHandler object processes only command events.

dispatchHandlerEvent

Calls the appropriate virtual function if a command event is received.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

A command handler contains event-processing members that you can use to process either an application or system command event. Override at least one of these virtual functions in a derived class.

command

Implemented by derived classes to handle application command events. You can use the function IEvent::window (p. 312) to reference the window to which the event is dispatched. Also, consider having your derived ICommandHandler class contain a data member that is a pointer to the window being handled.

```
virtual Boolean
  command( ICommandEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

systemCommand

Implemented by derived classes to handle the system command events.

You can check if the command is a predefined system command by comparing the value returned by ICommandEvent::commandId (p. 211) against the ISystemMenu item identifiers (p. 948).

```
virtual Boolean
  systemCommand( ICommandEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

ICommandHandler

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IContextHandle

Derivation
IBase
IHandle
IContextHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	219
operator _XtAppStruct *	219

The IContextHandle class accesses and manages the application context.

Motif IContextHandle is an alias for the X Toolkit type XtAppContext.

Public Functions

Constructors

You can construct objects of this class.

IContextHandle

You can construct objects of this class from an XtAppContext (a value of type IHandle::Value), which defaults to 0.

```
IContextHandle( _XtAppStruct* value = 0 );
```

Win	PM	Motif
N	N	Y

Operators

This operator allows you to use IContextHandle where an XtAppContext structure is expected.

operator _XtAppStruct *

Returns the IContextHandle value.

© Copyright IBM Corp. 1992, 1996

IContextHandle

```
operator _XtAppStruct *() const;
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

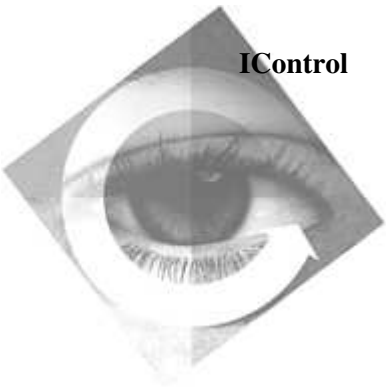
IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	



IControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl

Inherited By

IBaseListBox
IBaseSpinButton
ICanvas
IContainerControl
INotebook
IOutlineBox
IProgressIndicator
IScrollBar
ITextControl

Header File icontrol.hpp

Members

Member	Page	Member	Page
Constructor	224	group	224
disableGroup	222	isGroup	223
disableTabStop	222	isTabStop	223
enableGroup	222	tabStop	224
enableTabStop	223	~IControl	221

The IControl class is the abstract base class for classes representing window system controls, such as list boxes, entry fields, and push buttons.

AIX ignores the styles group and tabStop. The library uses the default Motif tab group behavior, instead, to maintain the Motif look-and-feel.

Public Functions

Constructors

You cannot construct objects of this class because it is an abstract base class. You can destruct objects of this class.

~IControl

```
virtual
  ~IControl();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Groups and Tab Stops

Use group and tab stop members to improve a user's navigation within a panel of controls. They are typically used on controls contained within canvases and dialogs.

disableGroup

Removes the group style from a control. When you remove the group style from a control, you actually merge two groups together. This is because when you define groups, a control must be in a group. For example, define five controls in three groups as follows:

Group 1	Control 1, Control 2
Group 2	Control 3
Group 3	Control 4, Control 5

When you remove the group style from a control, its group merges into the preceding group. Therefore, if you remove the group style from:

- Control 3, Group 2 merges into Group 1.
- Control 4, Group 3 merges into Group 2.
- Control 1, Group 1 merges into Group 3. This is because Group 1 has no group preceding it.

```
virtual IControl&
  disableGroup();
```

Win	PM	Motif
Y	*Y*	*I*

Motif The AIX release ignores the group style. For portable code, use IGroupBox instead. IGroupBox creates a dummy widget, which does not appear on the screen. If you make it the parent of any other control, the other control might not appear on the screen either. For portable applications, use ISetCanvas, which has functions that let you draw a box around the canvas and label your canvas. Or, you can use a combination of a canvas and an IOutlineBox control to simulate the IGroupBox behavior.

disableTabStop

Removes the tabStop style from a control, therefore preventing this control from being tabbed to.

```
virtual IControl&
  disableTabStop();
```

Win	PM	Motif
Y	*Y*	*I*

Motif AIX ignores the tabStop style.

enableGroup

Sets the group style of a control, therefore making this control first in a cursorable group of controls.

```
virtual IControl&
  enableGroup( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

Motif AIX ignores the group style.

enableTabStop

Sets the tabStop style of a control, therefore identifying this control as one that can be tabbed to.

```
virtual IControl&
  enableTabStop( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

Motif AIX ignores the tabStop style.

isGroup

If the control has the group style set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isGroup() const;
```

Win	PM	Motif
Y	Y	I

Motif AIX ignores the group style; therefore, this function returns false.

isTabStop

If the control has the tabStop style set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isTabStop() const;
```

Win	PM	Motif
Y	Y	I

Motif AIX ignores the tabStop style; therefore, this function returns false.

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Protected Functions

Constructors

You cannot construct objects of this class because it is an abstract base class. You can destruct objects of this class.

IControl

```
IControl();
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid control styles for the static member functions of the IControl (p. 221) class. You can use these styles with the styles in the following class:

IWindow Styles (p. 1093)

group Identifies the control as being the first in a group. No other controls in the group can have this style. Controls in the group must be siblings that you constructed following the first control. You can cursor through the resulting group and, when you reach the last control in the group, the cursor returns to the first control in the group.

```
static const Style
  group;
```

Win	PM	Motif
Y	Y	I

tabStop Identifies the control as one the user can tab to.

```
static const Style
  tabStop;
```

Win	PM	Motif
Y	Y	I

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IControl contains the following nested classes:

IControl::Style (see page 226)

© IBM Corp. 1992, 1996

IControl::Style

Derivation

IBase
IBitFlag
IControl::Style

Inherited By None.

Header File icontrol.hpp

The nested class IControl::Style provides a set of valid styles for the IControl (p. 221) class.

AIX ignores the styles group and tabStop. The library uses the default Motif tab group behavior instead to maintain the Motif look-and-feel. For portable code, use IGroupBox instead.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IControlEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent

Inherited By

ICnrEvent
ICustomButtonDrawEvent
IDrawItemEvent
IListBoxSizeItemEvent
IPageEvent

Header File ictlevt.hpp

Members

Member	Page
Constructor	228
controlId	228
~IControlEvent	228

The IControlEvent class represents a notification event from a control. Many control event handler classes, such as IEditHandler (p. 267), construct and process objects of this class. These notification events are first dispatched to the handlers attached to the control itself, then to the handlers attached to the owner window of the control.

PM An IControlEvent encapsulates the Presentation Manager message WM_CONTROL.

Motif An IControlEvent encapsulates various widget callbacks and certain X events.

Public Functions

Constructors

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

Typically, a handler object, in its dispatchHandlerEvent (p. 414) function, constructs an IControlEvent object from an IEvent (p. 304) object. The handler then passes the IControlEvent object to one of its virtual functions. For example, the class IEditHandler (p. 267) constructs an IControlEvent, which it passes to its edit (p. 269) function.

© Copyright IBM Corp. 1992, 1996

IControlEvent

IControlEvent

1 `IControlEvent( const IEvent& event );`

Win	PM	Motif
Y	Y	Y

Create an IControlEvent from a generic IEvent object.

2 `IControlEvent( const IControlEvent& event );`

Win	PM	Motif
Y	Y	Y

Create an IControlEvent from another IControlEvent object.

~IControlEvent

```
virtual
  ~IControlEvent();
```

Win	PM	Motif
Y	Y	Y

Event Information

Use these members to query information about the control that the event was generated from.

controlId Returns the window ID of the control the event applies to.

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

Win	PM	Motif
Y	Y	Y

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**	

© IBM Corp. 1992, 1996

ICoordinateSystem

Derivation

IBase
ICoordinateSystem

Inherited By None.

Header File icoordsy.hpp

Members

Member	Page	Member	Page
applicationOrientation	232	isConversionNeeded	232
convertToApplication	231	nativeOrientation	232
convertToNative	231	setApplicationOrientation	232

The ICoordinateSystem class maintains the settings information used to implement window coordinate mapping between the application coordinate system and the coordinate system used by the underlying GUI. Window coordinate mapping is needed because GUIs vary in their definition of the origin point of window coordinates. For example, Presentation Manager defines the 0,0 point of the screen to be the lower-left corner, and positive changes in the Y-coordinate move up the screen. Conversely, Motif and Windows define the 0,0 point to be the upper-left corner of the screen, and positive changes in the Y-coordinate move down the screen.

This class is used extensively by the User Interface Class Library implementation to map the coordinates used in an application to those required by the GUI toolkit. If you are writing application code that does not use native system calls, your primary use of this class will be to use ICoordinateSystem::setApplicationOrientation (p. 232) to specify your application's coordinate system. If you do not call this function, the default value of originLowerLeft (p. 233) is used.

If you are extending the library by writing classes for custom controls or providing other reusable classes based on User Interface Class Library, then consider using this class to provide coordinate-mapping for your users. Coordinate conversion is needed when you make calls to system routines that accept points or rectangles. You also may need it if you rely on a specific orientation to perform layout of windows or graphics.

The ICoordinateSystem class does not apply to the 2D graphic classes that are part of the User Interface Class Library.

© Copyright IBM Corp. 1992, 1996

Public Functions

Conversion

Conversion is the process of mapping a point or rectangle expressed in one orientation to one expressed in another orientation. To perform the conversion, a reference size is required. The *reference size* is the size of the coordinate space in which the point or rectangle to be converted is expressed. For example, the reference size of a top-level window positioned with respect to the screen size would be the screen size.

convertToApplication

Converts a point or rectangle from the native coordinate orientation to the application orientation.

1
```
static IPoint
  convertToApplication( const IPoint& nativePoint,
                        const ISize& referenceSize );
```

Win	PM	Motif
Y	Y	N

Returns a point in application coordinates. The point is computed from a point expressed in native orientation and a reference size. *referenceSize* should be the size of the coordinate space in which the native point coordinate is expressed.

2
```
static IRectangle
  convertToApplication( const IRectangle& nativeRectangle,
                        const ISize& referenceSize );
```

Win	PM	Motif
Y	Y	N

Returns a rectangle in application coordinates. The rectangle is computed from a rectangle expressed in native orientation and a reference size. *referenceSize* should be the size of the coordinate space in which the native point coordinate is expressed.

convertToNative

Converts a point or rectangle from the application coordinate orientation to the native orientation.

1
```
static IRectangle
  convertToNative( const IRectangle& applicationRectangle,
                   const ISize& referenceSize );
```

Win	PM	Motif
Y	Y	N

Returns a rectangle in native coordinates. The rectangle is computed from a rectangle expressed in application orientation and a reference size. *referenceSize* should be the size of the coordinate space in which the application point is expressed.

2
```
static IPoint
  convertToNative( const IPoint& applicationPoint,
                   const ISize& referenceSize );
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

Returns a point in native coordinates. The point is computed from a point expressed in application orientation and a reference size. *referenceSize* should be the size of the coordinate space in which the application point is expressed.

isConversionNeeded

Returns true if the native and application orientations are different. Otherwise, false is returned.

```
static Boolean
  isConversionNeeded();
```

Win	PM	Motif
Y	Y	N

Orientation

The orientation describes the reference point (0,0) used when a location in a window or screen is described.

applicationOrientation

Returns the currently set orientation to be used by the User Interface Class Library interface. Unless otherwise noted, all IRectangle and IPoint objects passed across the User Interface Class Library interface are in this orientation. For compatibility with prior releases, the default application orientation is originLowerLeft (p. 233).

```
static Orientation
  applicationOrientation();
```

Win	PM	Motif
Y	Y	N

nativeOrientation

Returns the orientation that is used natively on the current platform.

```
static Orientation
  nativeOrientation();
```

Win	PM	Motif
Y	Y	N

PM This function returns originLowerLeft (p. 233).

Win This function returns originUpperLeft (p. 233).

setApplicationOrientation

Sets the orientation to be used for the User Interface Class Library interface. If you want to change this value, you must call this function before any IWindow (p. 1044) objects are created.

Note: Calling this function after IWindow objects have been created may cause unpredictable results. If your design calls for using an orientation other than

the User Interface Class Library default of originLowerLeft (p. 233), set the orientation early in your program.

This function does not affect the processing of dialog templates loaded from resource files. The coordinates in resource files are always interpreted in native system coordinates when the dialog is being loaded.

```
static Orientation
  setApplicationOrientation(
   Orientation coordsystem = originLowerLeft);
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Orientation

```
Orientation {
    originUpperLeft,
    originLowerLeft
    };
```

Orientation is an enumeration describing the coordinate systems supported by User Interface Class Library. The *originUpperLeft* orientation has the origin located in the upper-left corner of the reference coordinate space. This is the orientation used by the native Windows and Motif toolkits. The *originLowerLeft* orientation has the origin located in the lower-left corner of the reference coordinate space. This is the coordinate system used by the Presentation Manager.

© IBM Corp. 1992, 1996

ICritSec

Derivation

IBase
ICritSec

Inherited By None.

Header File icritsec.hpp

Members

Member	Page
Constructor	235
count	235
~ICritSec	235

The ICritSec class isolates blocks of code you consider to be critical sections. Such blocks usually manipulate static data and need to do so without interruption from code that is executing in other threads.

Objects of this class guarantee that no other thread in the current process executes between the object's construction and destruction.

Typically, code that must execute without interruption by other threads is partitioned into a separate block, delimited by braces ({}). You create an object of this class within that block.

Note: While the critical section is executing, avoid actions having the potential to interrupt it, such as system calls. If you must preserve locks beyond the scope of a critical section object, use semaphores as implemented by the class IResourceLock (p. 820).

Motif Use this class to temporarily block signals from the current process. The constructor sets the process signal mask to block all maskable signals except SIGTRAP. SIGTRAP is not masked to allow debug tracing through a critical section. The destructor restores the process signal mask to the value it had before creation of the object.

Because signals are masked, some services might not operate correctly inside a critical section. An example of this is timer signals. If you attempt to lock an ISharedResource (p. 866) inside a critical section, any timeout you specify will be ineffective because the timer signal is masked.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

Use these members to construct and destruct objects of this class. The constructor and destructor implement all of the function of this class. The constructor causes a critical section to be entered. The corresponding destructor exits the critical section.

ICritSec Creates a critical section object and enters a critical section within the block of code where you create the object.

```
ICritSec();
```

Win	PM	Motif
Y	*Y*	*Y*

~ICritSec

Exits a critical section within the block of code where you create the critical section object and deletes the object.

```
~ICritSec();
```

Win	PM	Motif
Y	*Y*	*Y*

Diagnostics

Use these members for diagnostic purposes. They contain additional information on the critical section.

count Returns the total number of critical sections entered but not yet exited. It is equivalent to the number of ICritSec objects in existence.

```
static unsigned long
  count();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

ICritSec

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ICurrentApplication

Derivation

IBase
  IVBase
    IApplication
      ICurrentApplication

Inherited By None.

Header File iapp.hpp

Members

Member	Page	Member	Page
Constructor	241	run	240
argc	237	setArgs	238
argv	237	setResourceLibrary	239
asDebugInfo	238	setUserResourceLibrary	239
exit	240	userResourceLibrary	240
pib	241	~ICurrentApplication	241
resourceLibrary	238		

The ICurrentApplication class represents the program that is currently running. You are limited to a single object of this class. To obtain a reference to the object, use the static function IApplication::current (p. 48). The object of this class contains information that the User Interface Class Library maintains for each running application.

Public Functions

Arguments

Use these members to access or set the program arguments that can be passed to a program when it is executed.

argc Obtains the number of application arguments.

```
virtual int
  argc() const;
```

Win	PM	Motif
Y	Y	Y

argv Obtains the specified argument that is passed to the application.

```
virtual IString
  argv( int argumentNumber ) const;
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

argumentNumber
Integer value that represents the specified argument.

setArgs Sets the program arguments. Call this member function from your application's main function, passing its *argc* and *argv* argument values.

```
virtual ICurrentApplication&
  setArgs( int argc,
           const char * const argv [ ] );
```

Win	PM	Motif
Y	Y	Y

argc Integer value that states the number of arguments.
argv Pointer to an array of arguments.

Diagnostics

Use these members for diagnostic purposes. They return an IString representation of an object of this class.

asDebugInfo Use this function to return a diagnostic representation of the object. The information that is returned is composed of the following:

- User Interface Class Library resource library name
- User-defined resource library name
- Number of arguments
- Argument list

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

PM A pointer to the process information block is also returned as an unsigned long value.

Resource Libraries

Use these members to access the user's default application resource library and the User Interface Class Library resource library.

resourceLibrary

Obtains a reference to the User Interface Class Library's resource library. Resources required by the base library code are loaded from this resource library. If ICurrentApplication::setResourceLibrary (p. 239) has been called with the name of a dynamic link library (DLL), the handle of this library is returned. Otherwise, the environment variable *ICLUI RESLIB* is checked for the name of a DLL, and its handle is returned. You can set the environment variable using the statement *SET ICLUI RESLIB=myappdll* from an operating system command line.

```
virtual IResourceLibrary&
  resourceLibrary() const;
```

Win	PM	Motif
Y	*Y*	*Y*

PM If you call ICurrentApplication::setResourceLibrary (p. 239) with the name of a dynamic link library (DLL), the handle of this library is returned. Otherwise, the library checks the environment variable *ICLUI RESLIB* for the name of a DLL, and its handle is returned.

Set the environment variable by using the following:

```
SET ICLUI RESLIB=myappdll
```

If you have not set the environment variable, the default User Interface Class Library resource DLL, *cppoor3u.dll*, is used.

setResourceLibrary

Sets the resource library from which the User Interface Class Library's resources are loaded. You specify the resource library as the name of the dynamic link library (DLL) without the file extension. If you pass a 0 instead of a DLL name, the resources are loaded from the application's executable file.

Note: When you rename the User Interface Class Library resource DLL, cppoor3u.dll, to ship with your application (per the licensing agreement), call this function to specify the new resource DLL name.

```
virtual ICurrentApplication&
  setResourceLibrary( const char* resLibName );
```

Win	PM	Motif
Y	*Y*	*Y*

resLibName
Pointer to the resource library name.

PM You must specify the resource library as the name of the dynamic link library without the file extension. If you specify 0, the application loads the resources from the application's executable file.

Motif The AIX environment uses the standard X resource database; therefore, this function has no effect.

setUserResourceLibrary

Sets the user-defined resource library from which you obtain application resources. The default is the executable file.

Note: You can use an argument of 0 for the *resLibName* parameter to reset the user-defined resource library to the executable file.

© IBM Corp. 1992, 1996

```
virtual ICurrentApplication&
  setUserResourceLibrary( const char* resLibName );
```

Win	PM	Motif
Y	Y	Y

resLibName
Pointer to the resource library name.

PM The default is IResourceId (p. 808). If you specify 0 for *resLibName*, the application uses the executable file.

Motif The AIX environment uses the standard X resource database; therefore, this function has no effect.

userResourceLibrary

Obtains a reference to the default user-defined resource library for this application. The default is the executable file. If you do not explicitly specify a user-defined resource library via setUserResourceLibrary (p. 239), IResourceId (p. 808) uses the executable file.

```
virtual IResourceLibrary&
  userResourceLibrary() const;
```

Win	PM	Motif
Y	Y	Y

Starting and Stopping

Use these members to start or stop the current process.

exit

Ends the current thread of execution by calling ICurrentThread::exit (p. 244). If the current thread is thread 1, the process is ended.

```
virtual ICurrentApplication&
  exit();
```

Win	PM	Motif
Y	Y	Y

run

Starts the processing of events in the current process by calling ICurrentThread::processMsgs (p. 248).

```
virtual ICurrentApplication&
  run();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IApplication		
adjustPriority	asString	**currentPID**
asDebugInfo	**current**	id

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

The constructors of this class are protected to ensure that the static function IApplication::current (p. 48) returns the correct reference to the only object of this class. Also, the destructor is protected.

ICurrentApplication

You can only construct objects of this class with the default constructor, which does not require any arguments.

```
ICurrentApplication();
```

Win	PM	Motif
Y	*Y*	*Y*

~ICurrentApplication

```
virtual
  ~ICurrentApplication();
```

Win	PM	Motif
Y	*Y*	*Y*

Process Information

Use these members to access process information.

pib Returns a pointer to the current process' process information block.

```
struct pib_s&
  pib();
```

Win	PM	Motif
N	*Y*	*N*

© IBM Corp. 1992, 1996

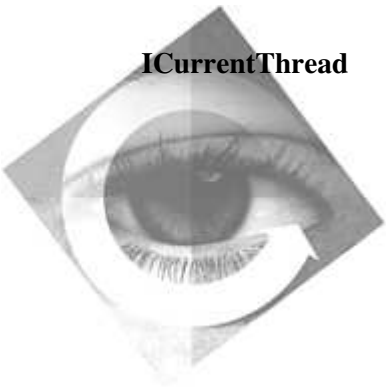
ICurrentApplication

Inherited Protected Functions

IApplication		
setId		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	



ICurrentThread

Derivation

IBase
  IVBase
    IThread
      ICurrentThread

Inherited By None.

Header File ithread.hpp

Members

Member	Page	Member	Page
Constructor	251	processMsgs	248
anchorBlock	246	remainingStack	244
appContext	246	setTopLevelShell	249
appShell	246	setXerrorCode	249
exit	244	sleep	245
handle	243	startedThread	251
id	244	suspend	250
initializeGUI	247	terminateGUI	248
isGUIInitialized	247	waitFor	245
isTopLevelShell	244	waitForAllThreads	245
isXerrorCodeAvailable	244	waitForAnyThread	246
messageQueue	248	XerrorCode	249

The ICurrentThread class represents the current thread of execution. An object of this class contains information that the User Interface Class Library maintains for the current thread of execution. You are limited to a single object of this class.

This class provides functions that you can only apply to the current thread of execution. To obtain a reference to the object, use the static function IThread::current (p. 995).

Public Functions

Current Thread Information

Use these members to access general information on the current thread.

handle Returns the thread handle for the current thread.

```
virtual IThreadHandle
  handle() const;
```

Win	PM	Motif
Y	*Y*	*I*

© Copyright IBM Corp. 1992, 1996

PM The thread handle and thread ID are identical.

Win You must use the thread handle for the operating system thread and synchronization functions.

id Obtains the current thread's identifier (ID).

```
virtual IThreadId
  id() const;
```

Win	PM	Motif
Y	Y	Y

Motif The AIX release of the User Interface Class Library supports only one thread. Therefore, this function returns the value 1.

Current Thread Support

Use these members to enable thread operations that only apply to the current thread.

exit Ends the current thread with the specified return value.

```
virtual void
  exit( unsigned long returnCode );
```

Win	PM	Motif
Y	Y	Y

returnCode

Unsigned long value that specifies the return value for termination.

isTopLevelShell

Determines if the application shell window is completely initialized. If it is, true is returned.

```
virtual Boolean
  isTopLevelShell() const;
```

Win	PM	Motif
N	N	Y

isXerrorCodeAvailable

Determines if an X Library error code is available for querying. If one is available, true is returned. This function uses XerrorCode to retrieve the error code, which resets this flag.

```
virtual Boolean
  isXerrorCodeAvailable() const;
```

Win	PM	Motif
N	N	Y

remainingStack

Obtains an approximation of the number of available bytes that remain on the stack. If the stack is not initialized, 0 is returned.

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

Win	PM	Motif
I	*Y*	*I*

Win This function always returns 1000 because an approximation cannot be obtained.

sleep

Suspends the thread for a specified number of milliseconds.

```
virtual ICurrentThread&
  sleep( unsigned long milliseconds );
```

Win	PM	Motif
Y	*Y*	*Y*

milliseconds
Unsigned long value that specifies the number of milliseconds to sleep.

Motif The granularity for sleeping on AIX is in seconds. The number of milliseconds is divided by 1000 to determine the number of seconds the process will sleep. If the number of milliseconds is less than 1 second, the process will not sleep.

waitFor

Allows the current thread to wait for a specified thread to end.

```
virtual ICurrentThread&
  waitFor( const IThread& anotherThread );
```

Win	PM	Motif
Y	*Y*	*I*

anotherThread
Reference to an IThread object that represents the specified thread.

W32s This member function has no effect.

Motif The AIX release of the User Interface Class Library supports a single-threaded environment; therefore, there are no other threads to wait for. This function immediately returns a reference to the ICurrentThread object.

Exceptions	
IInvalidParameter	The current thread did not wait for the specified thread to end. The specified thread is the current thread, which cannot wait on itself.
IAccessError	The current thread did not wait for the specified thread to end. The specified thread may have been invalid or an interrupt may have occurred.

waitForAllThreads

Allows the current thread to wait for all secondary threads to end.

```
virtual ICurrentThread&
  waitForAllThreads();
```

Win	PM	Motif
Y	*Y*	*I*

W32s This member function has no effect.

© IBM Corp. 1992, 1996

Motif The AIX release of the User Interface Class Library supports a single-threaded environment; therefore, there are no other threads to wait for. This function immediately returns a reference to the ICurrentThread object.

Exceptions	
IInvalidParameter	The current thread did not wait for all secondary threads to end. The current thread is not the primary thread.

waitForAnyThread

Allows the current thread to wait for the first termination to occur of any other threads in the current process. The terminated thread's identifier (ID) is returned.

```
virtual IThreadId
  waitForAnyThread();
```

Win	PM	Motif
Y	*Y*	*I*

W32s This member function has no effect.

Motif The AIX release of the User Interface Class Library supports a single-threaded environment; therefore, there are no other threads to wait for. This function immediately returns a reference to the ICurrentThread object.

Exceptions	
IAccessError	The current thread did not wait for the next thread in the process to terminate. No thread terminated or an interrupt may have occurred.

Graphical User Interface (GUI) Support

Use these members to support GUI activities on the current thread.

anchorBlock

Obtains the anchor block handle for the current thread. If the anchor block handle is not initialized, 0 is returned.

```
virtual IAnchorBlockHandle
  anchorBlock() const;
```

Win	PM	Motif
I	*Y*	*I*

Win This function always returns 1 because Windows has no concept of an anchor block.

appContext

Obtains the application context handle for the current thread. If the application context handle is not initialized, 0 is returned.

```
virtual IContextHandle
  appContext() const;
```

Win	PM	Motif
N	*N*	*Y*

appShell

Obtains the application shell handle for the current thread. If the application shell handle is not initialized, 0 is returned.

```
virtual IWindowHandle
  appShell() const;
```

Win	PM	Motif
N	N	Y

initializeGUI Initializes the graphical user interface (GUI) environment on the current thread.

```
virtual void
  initializeGUI( long queueSize = 30 );
```

Win	PM	Motif
Y	Y	Y

queueSize Long value that represents the queue size for the GUI environment. The User Interface Class Library default queue size is 30.

PM Sets up a Presentation Manager environment (anchor block and message queue).

Win The *queueSize* parameter has no effect because you can neither create a message queue nor specify a queue size under Windows.

W32s You can set the message queue size using this function. You must do so before the creation of any window classes, otherwise the function has no effect. It is recommended that the message queue size be set before calling any Windows functions.

If you do not call this function the queue size is set to 96, the recommended message queue size for applications using OLE. The User Interface Class Library drag and drop support uses OLE.

Motif Initializes the X Library environment by calling XtAppInitialize to establish a connection to the display. This function also creates an application context and an initial application shell.

Exceptions	
IAccessError	The graphical user interface (GUI) was not initialized. The window initialization failed.
IAccessError	The graphical user interface (GUI) was not initialized. The message queue creation failed.

isGUIInitialized

Determines if the graphical user interface (GUI) environment is active for this thread. If the GUI is active, true is returned.

```
virtual Boolean
  isGUIInitialized() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

ICurrentThread

messageQueue

Obtains the message queue handle for the current thread. A 0 is returned if the handle is not initialized or is invalid.

1
```
virtual IMessageQueueHandle
  messageQueue();
```

Win	PM	Motif
Y	Y	N

Use this version of the function if you want to reset an invalid message queue handle to 0. Do this if the handle is determined to be invalid.

Win Returns the current thread's identifier.

2
```
virtual IMessageQueueHandle
  messageQueue() const;
```

Win	PM	Motif
Y	Y	N

Use this version of the function if you want the invalid message queue handle to remain invalid.

processMsgs

Gets messages from the message queue and dispatches them to the appropriate handlers.

```
virtual void
  processMsgs();
```

Win	PM	Motif
Y	Y	Y

PM Messages are dispatched until the WM_QUIT message is received.

Motif ICurrentThread dispatches messages until it receives the client message WM_QUIT or the application is terminated through the Motif Window Manager.

terminateGUI

Terminates the graphical user interface (GUI) environment. If you start a thread with IThread::autoInitGUI (p. 986) and the function returns true, this function is called automatically by the User Interface Class Library once IThreadFn::run (p. 1005) is completed.

If you do the following, the User Interface Class Library does not call this function automatically, and it is your responsibility to do so:

- Start the thread with IThread::autoInitGUI and the function returns false
- Stop the thread using IThread::stop (p. 994)
- Stop the thread using ICurrentThread::exit (p. 244)

Note: If you do not call this function and it is your responsibility to do so, you can tie up system resources until your application ends.

```
virtual void
  terminateGUI();
```

Win	PM	Motif
I	*Y*	*Y*

PM Before calling the WinTerminate API, the application must destroy (that is delete) all windows and the message queue. This function destroys the message queue, but it is still the application's responsibility to destroy all of the window objects for the thread. If it does not, the return value of WinTerminate and any subsequent calls is indeterminate.

Win This function has no effect because the message queue is automatically destroyed by Windows.

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

setTopLevelShell

Called when the application shell window has been initialized. The User Interface Class Library calls this function to prevent multiple initializations of the graphical user interface.

Note: A user of the User Interface Class Library should never call this member function.

```
virtual void
  setTopLevelShell() const;
```

Win	PM	Motif
N	*N*	*Y*

setXerrorCode

Sets the error code that the last call to an X Library function reported. Typically, you use this function to save the error code detected by the handler set by XSetErrorHandler in IThread. The error handler is set by IThread so that the error codes are maintained on a per thread basis.

Note: A user of the User Interface Class Library should never call this member function.

```
virtual void
  setXerrorCode( int errorCode );
```

Win	PM	Motif
N	*N*	*Y*

errorCode Integer value that specifies the error code to set.

XerrorCode Returns the X Library error code set by the member function ICurrentThread::setXerrorCode (p. 249). You can retrieve the error code until a new X Library error code is set by ICurrentThread::setXerrorCode, but the flag indicating

© IBM Corp. 1992, 1996

that an error code is available is reset on the first invocation. The class IXLibErrorInfo uses this function to obtain information about the last X Library error code detected.

Note: A user of the User Interface Class Library should never call this member function.

```
virtual int
  XerrorCode() const;
```

Win	PM	Motif
N	*N*	*Y*

Suspending Threads

Use these members to suspend your current thread of execution if the thread is a non-GUI thread. Use IThread::resume (p. 992) to resume execution of the thread.

suspend Suspends the current thread of execution only if the graphical user interface (GUI) is not initialized for the thread. Suspending a GUI-initialized thread causes your application to hang.

```
virtual void
  suspend();
```

Win	PM	Motif
Y	*Y*	*I*

W32s This function does not suspend the current thread.

Motif You cannot suspend or resume AIX threads because the AIX release of the User Interface Class Library has a single-thread limitation.

Exceptions	
IInvalidRequest	The current thread was not suspended. A GUI thread cannot be suspended, or the application will hang.

Inherited Public Functions

IThread		
adjustPriority	id	setPriority
asDebugInfo	isStarted	setQueueSize
asString	messageQueue	setRelatedHandlesList
autoInitGUI	priorityClass	setStackSize
current	priorityLevel	setVariable
currentHandle	queueSize	setWindowList
currentId	relatedHandlesList	stackSize
defaultAutoInitGUI	resume	start

IThread		
defaultQueueSize	setAutoInitGUI	stop
defaultStackSize	**setDefaultAutoInitGUI**	stopProcessingMsgs
dialogControls	**setDefaultQueueSize**	suspend
handle	**setDefaultStackSize**	variable

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

Use these protected constructors to construct objects of this class.

ICurrentThread

The constructor is protected to prevent the accidental creation of objects of this class. Creation of objects of this class is restricted to IThread::current (p. 995).

```
ICurrentThread();
```

Win	PM	Motif
Y	*Y*	*Y*

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library.

startedThread

Returns a pointer to the object of the IStartedThread class that corresponds to the current thread.

```
virtual IStartedThread*
  startedThread() const;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

ICurrentThread

Inherited Protected Functions

IThread		
newStartedThread	operator =	startedThread

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IDeviceColor

Derivation

IBase
  IVBase
    IColor
      IDeviceColor

Inherited By None.

Header File icolor.hpp

Members

Member	Page
Constructor	253
deviceColor	253
~IDeviceColor	254

The IDeviceColor class works with device-independent color indexes.

Motif The color white is used for IDeviceColor::background. The color black is used for IDeviceColor::neutral and IDeviceColor::defaultColor.

Public Functions

Color Representation

Use these members to identify the device-independent color represented by an IDeviceColor object.

deviceColor Returns the DeviceColor (p. 254) enumerator value used to create this object.

```
DeviceColor
  deviceColor() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

IDeviceColor You can only construct objects of this class from a DeviceColor (p. 254) enumerator.

© Copyright IBM Corp. 1992, 1996

IDeviceColor

```
IDeviceColor( DeviceColor color );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	*color* is not a valid DeviceColor (p. 254) enumerator value, or the presentation system was unable to return the device color.

~IDeviceColor

```
virtual
  ~IDeviceColor();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IColor		
asPixel	index	redMix
asRGBLong	operator !=	setBlue
blueMix	operator =	setGreen
greenMix	operator ==	setRed

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

DeviceColor

```
DeviceColor {
    defaultColor,
    background,
    neutral
    };
```

Use these enumerators to specify the device-independent color of the object you are constructing:

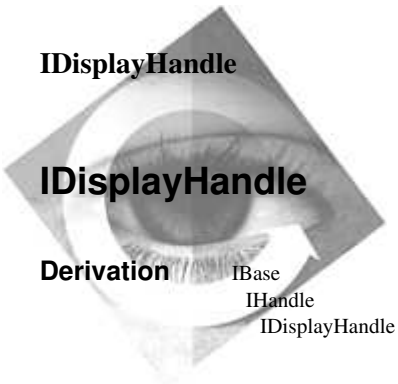
defaultColor
Specifies the default color of a device.

background
Specifies the background color of a device.

neutral
Specifies the neutral color of a device.

Motif The color values in Motif are hard coded. The background color is white, the neutral color is black, and the defaultColor is black.

© IBM Corp. 1992, 1996



IDisplayHandle

Derivation

IBase
IHandle
IDisplayHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	256
operator _XDisplay *	256

The IDisplayHandle class wrappers X-Toolkit objects of type Display *.

Motif IDisplayHandle is an alias for a pointer to the X Library type Display.

Public Functions

Constructors

You can construct objects of this class.

IDisplayHandle

Constructs objects of this class from a _XDisplay (a value of type IHandle::Value), which defaults to 0.

```
IDisplayHandle( _XDisplay* value = 0 );
```

Win	PM	Motif
N	*N*	*Y*

Operators

This operator allows you to use IDisplayHandle where an XDisplay structure is expected.

operator _XDisplay *

Returns the IDisplayHandle value.

© Copyright IBM Corp. 1992, 1996

```
operator _XDisplay *() 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**	

© IBM Corp. 1992, 1996

IDrawItemEvent

Derivation

IBase
IVBase
IEvent
IControlEvent
IDrawItemEvent

Inherited By

ICnrDrawItemEvent
IListBoxDrawItemEvent
IMenuDrawItemEvent
INotebookDrawItemEvent

Header File idievt.hpp

Members

Member	Page	Member	Page
Constructor	258	itemRect	259
itemId	259	ownerItemData	260
itemPresSpaceHandle	259	~IDrawItemEvent	259

The IDrawItemEvent class provides the draw-item event information for containers, list boxes, menus, notebooks, sliders, and progress indicators.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDrawItemEvent

Constructs an IDrawItemEvent from the specified event. ISliderDrawHandler::dispatchHandlerEvent (p. 894) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to its event-handling functions.

Classes derived form IDrawItemEvent also call this constructor.

```
IDrawItemEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

~IDrawItemEvent

```
virtual
  ~IDrawItemEvent();
```

Win	PM	Motif
Y	Y	Y

Item Information

The event contains information for drawing an item. Use these members to identify the item and its location.

itemId Returns the identifier for the item.

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

Win	PM	Motif
Y	Y	Y

itemPresSpaceHandle

Returns the presentation space to use for drawing the item.

```
virtual IPresSpaceHandle
  itemPresSpaceHandle() const;
```

Win	PM	Motif
Y	Y	Y

itemRect Returns the screen rectangle of the item, relative to the control returned by IEvent::controlWindow (p. 310).

```
virtual IRectangle
  itemRect() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IControlEvent		
controlId		

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

© IBM Corp. 1992, 1996

IDrawItemEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Implementation

These members access the draw-item data from objects of this class.

ownerItemData

Returns a pointer to a structure holding the draw-item data.

```
void*
  ownerItemData() const;
```

Win	PM	Motif
Y	*Y*	*N*

PM The returned value is a pointer to an *OWNERITEM* structure.

Win The returned value is a pointer to a *DRAWITEMSTRUCT* structure.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IDynamicLinkLibrary

Derivation

IBase
  IVBase
    IResourceLibrary
      IDynamicLinkLibrary

Inherited By None.

Header File ireslib.hpp

Members

Member	Page	Member	Page
Constructor	261	isEntryPoint32Bit	264
asDebugInfo	263	isOpen	263
asString	263	open	264
close	264	operator =	262
fileName	263	procAddress	265
handle	263	~IDynamicLinkLibrary	262

The IDynamicLinkLibrary class supports the loading of resources from a dynamic link library (DLL).

Motif The AIX release of the User Interface Class Library does not support this class.

Public Functions

Constructors

You can construct, copy, assign, and destruct objects of this class. You can construct objects of this class using either a dynamic link library name, module handle, or a reference to an existing object of this class.

IDynamicLinkLibrary

1 `IDynamicLinkLibrary( const IModuleHandle& moduleHandle );`

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

moduleHandle
Reference to a module handle of a DLL that has been loaded explicitly.

Use this version of the constructor if you have an IModuleHandle (p. 618) obtained by loading the dynamic link library (DLL) explicitly.

Exceptions	
IAccessError	The dynamic link library object was not created. The module handle may be invalid.

2 `IDynamicLinkLibrary( const char* resourceFileName );`

Win	PM	Motif
Y	*Y*	*N*

resourceFileName
Pointer to the dynamic link library name.

Use this version of the constructor if you know the name of the dynamic link library (DLL).

Note: If you want to search the library path for the DLL, specify *resourceFileName* without the path or extension.

3 `IDynamicLinkLibrary( const IDynamicLinkLibrary& dllLibrary );`

Win	PM	Motif
Y	*Y*	*N*

dllLibrary Reference to an existing dynamic link library object.

Creates a dynamic link library object using a reference to an existing dynamic link library object. This is commonly known as a copy constructor.

operator =

Assigns the member data of an object of this class to another object of this class.

```
IDynamicLinkLibrary&
  operator =( const IDynamicLinkLibrary& dllLibrary );
```

Win	PM	Motif
Y	*Y*	*N*

dllLibrary Reference to an existing dynamic link library object.

~IDynamicLinkLibrary

The dynamic link library is unloaded from memory if it is no longer being referenced.

```
virtual
  ~IDynamicLinkLibrary();
```

Win	PM	Motif
Y	*Y*	*N*

Diagnostics

Use these members for diagnostic purposes. They return an IString (Vol. I) representation of an object of this class.

asDebugInfo Provides textual information about the class object. It returns a string that contains the file name of the dynamic link library.

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	N

asString Provides textual information about the class object. It returns a string that contains the file name of the dynamic link library.

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	N

Dynamic Link Library Information

Use these members to query general dynamic link library information.

fileName Returns the fully qualified file name of the dynamic link library.

```
virtual IString
  fileName() const;
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The file name was not returned. The module handle may be invalid.

handle Returns the module handle of the dynamic link library.

```
virtual IModuleHandle
  handle() const;
```

Win	PM	Motif
Y	Y	N

isOpen Returns true if the dynamic link library is open.

```
virtual Boolean
  isOpen() const;
```

Win	PM	Motif
Y	Y	N

Opening and Closing

Use these members to open or close a dynamic link library (DLL).

© IBM Corp. 1992, 1996

Note: DLL objects are reference-counted. When a DLL is opened, the reference count is incremented, and when it is closed, the reference count is decremented. When the reference count reaches 0, the DLL is unloaded from memory.

close

Closes a dynamic link library and decrements the reference count.

```
virtual IDynamicLinkLibrary&
  close();
```

Win	PM	Motif
Y	Y	N

open

Opens a dynamic link library and increments the reference count.

```
virtual IDynamicLinkLibrary&
  open();
```

Win	PM	Motif
Y	Y	N

Procedure Address Loading

Use these members to get a procedure address from a dynamic link library or to test the memory model of an entry point.

isEntryPoint32Bit

If the entry point is a 32-bit function, true is returned.

1
```
Boolean
  isEntryPoint32Bit( const char* procedureName ) const;
```

Win	PM	Motif
Y	Y	N

procedureName
Pointer to the procedure name.

Use this version of the function if you know the entry point's name.

Exceptions	
IAccessError	The entry point was not queried. The procedure name may be invalid.

2
```
Boolean
  isEntryPoint32Bit( unsigned long procedureOrdinal ) const;
```

Win	PM	Motif
Y	Y	N

procedureOrdinal
Unsigned long value that represents the procedure ordinal.

Use this version of the function if you know the entry point's ordinal.

Exceptions	
IAccessError	The entry point was not queried. The procedure ordinal may be invalid.

procAddress Loads the specified procedure address from a dynamic link library.

1

```
void*
  procAddress( const char* procedureName ) const;
```

Win	PM	Motif
Y	*Y*	*N*

procedureName
Pointer to the procedure name.

Use this version of the function if you know the procedure name.

Exceptions	
IAccessError	The procedure address was not loaded. The procedure name may be invalid.

2

```
void*
  procAddress( unsigned long procedureOrdinal ) const;
```

Win	PM	Motif
Y	*Y*	*N*

procedureOrdinal
Unsigned long value that represents the procedure ordinal.

Use this version of the function if you know the procedure ordinal.

Exceptions	
IAccessError	The procedure address was not loaded. The procedure ordinal may be invalid.

Inherited Public Functions

IResourceLibrary		
asDebugInfo	loadBitmap	loadPointer
asString	loadDialog	loadString
fileName	loadHelpTable	operator =
handle	loadIcon	**sizeBitmapTo**
isOpen	loadMenu	tryToLoadBitmap
loadAccelTable	loadMessage	tryToLoadIcon

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IDynamicLinkLibrary

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IEditHandler

Derivation

IBase
  IVBase
    IHandler
      IEditHandler

Inherited By None.

Header File iedithdr.hpp

Members

Member	Page	Member	Page
Constructor	269	edit	269
dispatchHandlerEvent	269	~IEditHandler	268

The IEditHandler class handles events resulting from a user changing a control's input value, such as changing the value of an entry field or moving the arm of a slider. IEditHandler objects process edit events for the following controls:

- ICircularSlider (Vol. III)
- IComboBox (p. 182)
- IEntryField (p. 271)
- IMultiLineEdit (p. 641)
- INumericSpinButton (p. 672)
- IProgressIndicator (p. 743)
- ISlider (p. 874)
- ITextSpinButton (p. 963)

You create a handler derived from IEditHandler and attach it to either the control whose input users can change or to the control's owner window. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the appropriate control window or owner window to the edit handler.

When the edit handler receives an edit event, it creates an IControlEvent (p. 227) object and routes that object to the IEditHandler::edit (p. 269) virtual function. You override this virtual function to supply your own specialized processing of an edit event.

The following return values from the virtual function specify whether the control event is passed on for additional processing:

© Copyright IBM Corp. 1992, 1996

true The edit event requires no additional processing. Do not pass it to another handler.

false The edit event requires additional processing. Pass the edit event to the next handler, as follows:

- If there is another handler for the control, pass the edit event to the next handler.
- If this is the last handler for the control, call IWindow::dispatch (p. 1082) to dispatch the edit event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the edit event.

Public Functions

Constructors

Only derived classes can construct objects of this class.

~IEditHandler

```
virtual
  ~IEditHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

Only derived classes can construct objects of this class.

IEditHandler Derived classes call this default constructor to create objects of this class.

```
IEditHandler();
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IEditHandler object processes only events related to changes to a window's value.

dispatchHandlerEvent

If an edit event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

An edit handler allows you to process changes to a window's value through its event-processing members. Override at least one of these virtual functions in a derived class.

edit Implemented by the derived classes to process input changes to a window.

```
virtual Boolean
  edit( IControlEvent& event ) = 0;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

© IBM Corp. 1992, 1996

IEditHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IEntryField

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IEntryField

Inherited By

IBaseComboBox

Header File

ientryfd.hpp

Members

Member	Page	Member	Page
Constructor	278	enableDataUpdate	282
alignment	272	enableInsertMode	280
anyData	291	enableMargin	281
autoScroll	291	enableNotification	281
autoTab	292	end	291
backgroundColor	277	foregroundColor	277
border3D	292	hasSelectedText	285
calcMinimumSize	289	hasTextChanged	285
centerAlign	292	initialize	288
characterTypeId	290	insertModeId	291
charType	273	isAutoScroll	274
classDefaultStyle	292	isAutoTab	274
clear	275	isCommand	278
command	292	isDragStarting	288
convertToGUIStyle	284	isEmpty	285
copy	275	isInsertMode	280
cursorPosition	282	isMargin	281
cut	276	isWriteable	282
dataUpdateId	290	leftAlign	293
dbcsData	292	leftIndex	280
defaultStyle	284	limit	286
disable	281	limitId	291
disableAutoScroll	274	lowerCase	293
disableAutoTab	274	margin	293
disableCommand	278	mixedData	293
disableDataUpdate	282	moveSizeTo	287
disableInsertMode	279	passEventToOwner	288
disableMargin	280	paste	277
discard	276	readOnly	293
enable	282	registerCallbacks	289
enableAutoScroll	274	removeAll	286
enableAutoTab	274	resetTextChangedFlag	285
enableCommand	278	rightAlign	293

© Copyright IBM Corp. 1992, 1996

IEntryField

Member	Page	Member	Page
sbcsData	293	setLeftIndex	280
selectedRange	283	setLimit	286
selectedText	283	setStyle	289
selectedTextLength	283	setTextChangedFlag	285
selectRange	283	text	282
setAlignment	273	unreadable	294
setCharType	273	unregisterCallbacks	289
setCursorPosition	283	upperCase	294
setDefaultStyle	284	~IEntryField	279
setLayoutDistorted	290		

The IEntryField class creates and manages entry field controls.

You derive classes from the following handlers and attach them to an entry field control:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- IScrollHandler (p. 845)

Motif The User Interface Class Library implements this class with the XmText widget (instead of XmTextField) to allow more uniformity between IEntryField and IMultiLineEdit.

You can implement specific keystroke processing, such as creating a numeric-only input field, by deriving a handler from IEditVerifyHandler and attaching your handler to the entry field object.

Public Functions

Attributes

Use these members to query and change characteristics of the entry field control.

alignment Returns the current alignment for this entry field object. The returned value is an enumerator provided by Alignment (p. 295).

```
Alignment
  alignment() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif In AIX, this function always returns a left alignment.

charType Returns the current type of character that this entry field accepts. The returned value is an enumerator provided by CharType (p. 295).

```
CharType
  charType() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif In Motif, an IEntryField is always of type *any*.

setAlignment Sets the alignment of the entry field control.

alignment Use the enumeration Alignment (p. 295) to specify the alignment of the entry field.

```
virtual IEntryField&
  setAlignment( Alignment alignment );
```

Win	PM	Motif
I	*Y*	*Y*

Win This member has no effect on Windows. Alignment can be specified on the constructor, but cannot be changed dynamically.

Motif In AIX, you can only set the alignment to left.

Exceptions	
IInvalidParameter	An invalid Alignment was specified. You must specify one of the valid Alignment values. Current valid values are left, center, and right.

setCharType Sets the type of character the entry field control can accept.

type Use the enumeration CharType (p. 295) to specify the type of characters in the entry field.

```
virtual IEntryField&
  setCharType( CharType type );
```

Win	PM	Motif
I	*Y*	*Y*

Motif AIX only supports the CharType enumerator *any*. This type corresponds to the X multibyte character set (MBCS).

Exceptions	
IInvalidParameter	An invalid CharType was specified. You must specify one of the valid CharType values. Current valid values are sbcs, dbcs, any, and mixed.

Auto Scroll

Use auto scroll members to query and change the autoScroll style of the entry field object. The autoScroll style determines if the entry field automatically scrolls in the appropriate direction if the user tries to move off the end of a line.

© IBM Corp. 1992, 1996

disableAutoScroll

Disables the style autoScroll (p. 291) on an entry field control.

```
virtual IEntryField&
  disableAutoScroll();
```

Win	PM	Motif
I	*Y*	*Y*

enableAutoScroll

Enables or disables the style autoScroll (p. 291) on an entry field control.

```
virtual IEntryField&
  enableAutoScroll( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

isAutoScroll

If the entry field control has the style autoScroll (p. 291) set, true is returned. Otherwise, false is returned.

```
Boolean
  isAutoScroll() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Auto Tab

Use auto tab members to query and change the autoTab style of the entry field object. The autoTab style determines if the entry field will automatically generate a tab character when the field is filled by adding a character to the end of the text.

disableAutoTab

Disables the style autoTab (p. 292) on an entry field control.

```
virtual IEntryField&
  disableAutoTab();
```

Win	PM	Motif
I	*Y*	*Y*

enableAutoTab

Enables or disables the style autoTab (p. 292) on an entry field control.

```
virtual IEntryField&
  enableAutoTab( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

isAutoTab

If the entry field control has the style autoTab (p. 292) set, true is returned. Otherwise, false is returned.

```
Boolean
  isAutoTab() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Clipboard Operations

Use these members to transfer data between the clipboard and the entry field control. Each of these operations deals with the selected text in the entry field control.

clear Replaces the selected text in the entry field with blanks.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the entry field is selected before making the call to clear.

To remove the text contents of the entry field, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IEntryField&
  clear( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Exceptions	
IInvalidRequest	The entry field has no text selected.

copy Copies the selected text to the clipboard.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the entry field is selected before making the call to copy.

```
virtual IEntryField&
  copy( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidRequest	The entry field has no text selected.
IAccessError	The operating system's request to copy from the entry field failed. See the exception text for further information about the failure.

cut Removes the selected text from the entry field control and puts it in the clipboard.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the entry field is selected before making the call to cut.

To remove the text contents of the entry field, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IEntryField&
  cut( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Exceptions	
IInvalidRequest	The entry field has no text selected.
IAccessError	The operating system's request to cut from the entry field failed. See the exception text for further information about the failure.

discard Deletes the selected text.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the entry field is selected before making the call to discard.

To remove the text contents of the entry field, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IEntryField&
  discard();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The entry field has no text selected.
IAccessError	The operating system's request to discard the text from the entry field failed. See the exception text for further information about the failure.

paste

Copies text from the clipboard to the entry field control, replacing any selected text.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the entry field is selected before making the call to paste.

```
virtual IEntryField&
  paste();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The clipboard does not contain any text.
IAccessError	The operating system's request to paste to the entry field failed. See the exception text for further information about the failure.

Colors

Use these members to query, set, and reset colors of the entry field.

backgroundColor

Returns the background color value of the entry field. If you have not set this color, the default is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	Y	N

This member is overridden in this derived class for specific operating system behavior.

foregroundColor

Returns the foreground color value of the entry field. If you have not set this color, the default is returned.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	Y	N

This member is overridden in this derived class for specific operating system behavior.

© IBM Corp. 1992, 1996

Commands

Use these members to query and change the command style of the entry field object. The command style identifies the entry field as a command entry field. This information can be used to provide command help if the user requests help for this field.

disableCommand

Disables the style command (p. 292) on an entry field control.

```
virtual IEntryField&
  disableCommand();
```

Win	PM	Motif
I	Y	Y

enableCommand

Enables or disables the style command (p. 292) on an entry field control.

```
virtual IEntryField&
  enableCommand( Boolean enable = true );
```

Win	PM	Motif
I	Y	Y

isCommand

If the entry field control has the style command (p. 292) set, true is returned. Otherwise, false is returned.

```
Boolean
  isCommand() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

IEntryField

1

```
IEntryField( unsigned long id,
             IWindow* parent,
             IWindow* owner,
             const IRectangle& initial = IRectangle ( ),
             const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the parent window, owner window, optional size and location, and optional style parameters.

id An entry field control ID.

parent The parent window.

owner The owner window.

initial A rectangle for the control. It specifies the initial position and size of the object you are constructing. Optional.

style The initial style for the control. The default is classDefaultStyle (p. 292). Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2
```
IEntryField( unsigned long id,
             IWindow* parent );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the parent window.

id An entry field control ID.
parent The parent window.

3
```
IEntryField( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the handle of an existing entry field.

handle The window handle of an existing entry field control.

~IEntryField

```
virtual
  ~IEntryField();
```

Win	PM	Motif
Y	Y	Y

Insert Mode

Use these members to set and query the insert mode for the entry field. The insert mode determines whether characters are inserted or replaced at the cursor position.

disableInsertMode

Enables the entry field for overtype mode and changes the cursor's appearance.

```
virtual IEntryField&
  disableInsertMode();
```

Win	PM	Motif
I	Y	Y

© IBM Corp. 1992, 1996

enableInsertMode

Enables the entry field for insert mode and changes the cursor's appearance.

```
virtual IEntryField&
  enableInsertMode( Boolean insert = true );
```

Win	PM	Motif
I	Y	Y

isInsertMode

Queries whether the entry field is in insert mode.

```
Boolean
  isInsertMode() const;
```

Win	PM	Motif
Y	Y	Y

Left Index

Use these members to set and query the left index of the entry field. The left index is the index of the first displayed character at the left edge of the entry field.

leftIndex

Returns the index of the first character displayed at the left edge of the entry field control. The index is 0-based.

```
unsigned long
  leftIndex() const;
```

Win	PM	Motif
I	Y	Y

Win This member is not supported on Windows. It will always return 0 on this platform.

setLeftIndex

Sets the first displayed character at the left edge of the entry field control. The index is 0-based.

```
virtual IEntryField&
  setLeftIndex( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to set the left index failed. See the exception text for further information about the failure.

Margins

Use margin members to query and change the margin style of the entry field object. The margin style determines if the entry field control will be drawn with a border around the editable text.

disableMargin

Disables the style margin (p. 293) on an entry field control. As a result, the entry field will not have a border surrounding its edit rectangle.

```
virtual IEntryField&
  disableMargin();
```

Win	PM	Motif
I	*Y*	*Y*

enableMargin

Enables or disables the style margin (p. 293) on an entry field control. As a result, this function adds or removes a border from the edit area.

```
virtual IEntryField&
  enableMargin( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

isMargin

If the entry field control has the style margin (p. 293) set, true is returned. Otherwise, false is returned.

```
Boolean
  isMargin() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables or disables the entry field control to send notifications to any observer objects.

```
virtual IEntryField&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*Y*

Overrides

These members are overloaded to ensure a disabled entry field is set to a read-only state to prevent user input.

disable

Disables an entry field control. When you disable the entry field, the entry field has the following characteristics:

- No longer accepts a keyboard event
- No longer accepts a mouse event
- Is grayed out
- Beeps when the user tries to enter data into the entry field

```
virtual IEntryField&
  disable();
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

enable Enables or disables an entry field control.

```
virtual IEntryField&
  enable( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

text Returns the text of the entry field's control window.

```
virtual IString
  text() const;
```

Win	PM	Motif
N	N	Y

Motif This override is added for a Motif-unreadable style.

Read-Only Operations

These members query and modify the read-only mode in the entry field, which specifies if the user can change the entry field text.

disableDataUpdate

Prevents inserting or changing characters in the entry field's text.

```
virtual IEntryField&
  disableDataUpdate();
```

Win	PM	Motif
Y	Y	Y

enableDataUpdate

Enables or disables the read-only mode on an entry field control.

```
virtual IEntryField&
  enableDataUpdate( Boolean update = true );
```

Win	PM	Motif
Y	Y	Y

isWriteable If the contents of the entry field can be modified, true is returned. Otherwise, false is returned.

```
Boolean
  isWriteable() const;
```

Win	PM	Motif
Y	Y	I

Selected Text

Use these members to manipulate selected text.

cursorPosition

Returns the character position from the start of the entry field to the current cursor location.

```
unsigned long
  cursorPosition() const;
```

Win	PM	Motif
Y	Y	N

selectedRange

Returns the range of the selected text. If no text is selected, an exception is thrown. The selected range is the 0-based index of the first character selected and the 0-based index of the last character selected.

```
IRange
  selectedRange() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The entry field has no text selected.

selectedText

Returns the selected text string.

```
IString
  selectedText() const;
```

Win	PM	Motif
Y	Y	Y

selectedTextLength

Returns the size of the selected area in bytes. The length does not include the NULL terminator. No exception is thrown if there is no text selected; 0 is returned.

```
unsigned long
  selectedTextLength() const;
```

Win	PM	Motif
Y	Y	Y

selectRange

Selects a range of text.

range The range is the index of the first character selected and the index of the last character selected. The index is 0-based. If you do not specify *range*, the default selects all of the text. If you specify a range with equivalent lower and upper values (n, n), the selection becomes an insertion point.

```
virtual IEntryField&
  selectRange( const IRange& range = IRange ( 0 , end ),
               unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

setCursorPosition

Moves the cursor to a specific position in the entry field. The count begins at the first position in the entry field, not at the cursor's current position.

© IBM Corp. 1992, 1996

```
virtual IEntryField&
  setCursorPosition( unsigned long newCursorPos );
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The operating system's request to set the cursor position failed. See the exception text for further information about the failure.

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the entry field styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 292) unless you have changed the style using setDefaultStyle (p. 284).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent entry fields.

style Use the styles provided by IEntryField Styles (p. 290) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Test Operations

Use these members to query the entry field.

hasSelectedText

If any of the entry field text is selected, true is returned. Otherwise, false is returned.

```
Boolean
  hasSelectedText() const;
```

Win	PM	Motif
Y	Y	Y

hasTextChanged

If any changes have been made to the entry field since the last time the changed flag was reset, true is returned. Otherwise, false is returned.

```
Boolean
  hasTextChanged() const;
```

Win	PM	Motif
Y	Y	Y

isEmpty

If the entry field is empty, true is returned. Otherwise, false is returned.

```
Boolean
  isEmpty() const;
```

Win	PM	Motif
Y	Y	Y

resetTextChangedFlag

Resets the changed flag so that from this point forward changes to the entry field can be detected. This is the same as IEntryField::setTextChangedFlag(false).

```
virtual IEntryField&
  resetTextChangedFlag();
```

Win	PM	Motif
Y	Y	I

setTextChangedFlag

Sets a flag indicating the changed status of the entry field. If changed=true, a flag is set to indicate the entry field contents have changed. if changed=false, a flag is set to indicate the entry field contents have not changed.

```
virtual IEntryField&
  setTextChangedFlag( Boolean changed = true );
```

Win	PM	Motif
Y	Y	I

Text Limit

Use these members to set and query the text limit of the entry field.

© IBM Corp. 1992, 1996

limit Returns the length, in bytes, of the longest text the entry field can hold.

Note: The default value for the limit differs from system to system in accordance with the look-and-feel of that system. If your application requires a specific limit for an entry field, you must set that limit using setLimit to ensure portability of your application.

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

Win	PM	Motif
Y	Y	Y

PM The default text limit in the OS/2 operating system environment is 32.

Exceptions	
IAccessError	The operating system's request to query the entry field limit failed. See the exception text for further information about the failure.

setLimit Sets the maximum number of bytes the entry field can contain.

The calcMinimumSize function uses the text limit to determine the minimum size of the entry field.

Note: The default value for the limit differs from system to system in accordance with the look-and-feel of that system. If your application requires a specific limit for an entry field, you must set that limit using setLimit to ensure portability of your application.

1
```
virtual IEntryField&
  setLimit( unsigned long textLimit );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The entry field already contains more characters than the proposed limit.

2
```
virtual IEntryField&
  setLimit( const IResourceId& textLimit );
```

Win	PM	Motif
Y	Y	Y

PM The default text limit in the OS/2 operating system environment is 32.

Text Removal

These members remove text from the entry field.

removeAll Deletes the entire contents of the entry field control.

```
virtual IEntryField&
  removeAll();
```

Win	PM	Motif
Y	Y	Y

Window Positioning

Use these members to set and query the size and position of windows.

moveSizeTo Changes the position and size of the entry field window using a coordinate system that is lower-left origin-based.

aRectangle A rectangle for the entry field window.

```
virtual IEntryField&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

© IBM Corp. 1992, 1996

Constructors

You can construct and destruct objects of this class.

IEntryField

```
IEntryField();
```

Win	PM	Motif
Y	Y	Y

Constructs objects of this class. This is a protected constructor that can be used by derived classes. This is the default constructor and accepts no parameters.

Drag and Drop Support

Use these members to customize the conditions for detecting a drag operation.

isDragStarting

Request the start of a drag operation. A drag operation will occur if the entryfield contains selected text and the mouse pointer was over that selection area when the button was pressed.

```
virtual Boolean
  isDragStarting( IEvent& event );
```

Win	PM	Motif
Y	N	N

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

initialize

Initializes private data for the entry field control. Both the IEntryField constructors and the derived classes' constructors call this function. This function can only be called after IWindow::startHandlingEventsFor (p. 1084) or IWindow::create (p. 1086) has been called.

```
IEntryField&
  initialize( unsigned long style );
```

Win	PM	Motif
N	N	Y

passEventToOwner

Determines whether an event is passed to the owner of this control.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	N	Y

registerCallbacks

Registers all possible callbacks to this object for events it might receive. IHandler-derived classes later determine which events they will process.

This class does not add any X event handlers.

If classes you derive override the registerCallbacks member function, the override must call Inherited::registerCallbacks to register the applicable callbacks.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes any callbacks and X event handlers added by the registerCallbacks function.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Implementation

These members provide utilities used to implement this class.

setStyle

Replaces the style of an entry field control, while preserving the cursor position and selected text.

```
virtual IEntryField&
  setStyle( unsigned long style );
```

Win	PM	Motif
Y	*Y*	*N*

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size that this entry field control can be, based on the text string length limit and the current font.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

setLayoutDistorted

Indicates that changes have occurred in the window causing the layout of the window in a canvas to be updated.

```
virtual IEntryField&
  setLayoutDistorted( unsigned long layoutAttributeOn,
                      unsigned long layoutAttributeOff );
```

Win	PM	Motif
N	*N*	*Y*

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

characterTypeId

Notification identifier provided to observers when the character type of the entry field control changes. IEntryField provides the character type in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is one of the predefined CharType enum values defined in this class.

```
static INotificationId const
  characterTypeId;
```

Win	PM	Motif
Y	*Y*	*Y*

dataUpdateId

Notification identifier provided to observers when the data update mode of the entry field control changes. IEntryField provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the user can now update the data, and false if the user cannot update the data.

```
static INotificationId const
  dataUpdateId;
```

Win	PM	Motif
Y	*Y*	*Y*

insertModeId Notification identifier provided to observers when the insert mode of the entry field control changes. IEntryField provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if insert mode is now enabled, and false if replace mode is enabled.

```
static INotificationId const
  insertModeId;
```

Win	PM	Motif
I	*Y*	*Y*

limitId Notification identifier provided to observers when the text limit of the entry field control changes. IEntryField provides the new text limit value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value will be the new text limit.

```
static INotificationId const
  limitId;
```

Win	PM	Motif
Y	*Y*	*Y*

Selected Text

Use these members to manipulate selected text.

end Denotes the end of the text for selecting text.

```
static const long
  end;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the entry field styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

anyData Sets the entry field to accept text that is a mixture of single-byte character set (SBCS) and double-byte character set (DBCS) characters.

If the text contains both SBCS and DBCS characters and will be converted from an ASCII code page to an EBCDIC code page, this style causes an entry field to ignore accounting for shift-in and shift-out characters that would be introduced into its text.

```
static const Style
  anyData;
```

Win	PM	Motif
Y	*Y*	*Y*

autoScroll If the user tries to move off the end of a line, the entry field automatically scrolls one-third the width of the window in the appropriate direction.

© IBM Corp. 1992, 1996

```
static const Style
  autoScroll;
```

Win	PM	Motif
Y	Y	Y

autoTab Causes a tab key to be generated when the entry field is filled by adding a character at the text limit of the entry field text.

```
static const Style
  autoTab;
```

Win	PM	Motif
I	Y	Y

border3D Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	I	I

Win This style is ignored on Windows NT 3.51 with Program Manager.

Win32s This style is ignored on Win32s.

centerAlign Sets the text in the entry field to be centered.

```
static const Style
  centerAlign;
```

Win	PM	Motif
Y	Y	Y

Win This style cannot be mixed with the autoScroll or unreadable styles.

classDefaultStyle

Provides the original default style for this class, which is the following: IWindow::visible | IEntryField::margin | IEntryField::autoScroll | IEntryField::leftAlign | IEntryField::anyData | IEntryField::border3D.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

command Creates information used by the Help Manager to provide command help if the user requests help for this entry field. No more than one entry field on each window should be given this style.

```
static const Style
  command;
```

Win	PM	Motif
I	Y	Y

dbcsData Sets the entry field to accept double-byte text only.

```
static const Style
  dbcsData;
```

Win	PM	Motif
I	*Y*	*Y*

leftAlign Set the text in the entry field to be left-justified.

```
static const Style
  leftAlign;
```

Win	PM	Motif
Y	*Y*	*Y*

lowerCase Displays the text in the entry field in lowercase characters.

```
static const Style
  lowerCase;
```

Win	PM	Motif
Y	*N*	*N*

margin Causes a border to be drawn around the entry field, with a margin between the border and the entry field. The margin's size is determined by the current font being used for entry field text (half a character-width wide and half a character-height high).

Because the margin and border are drawn around the entry field, using the margin style does not change the position of the entry field itself.

```
static const Style
  margin;
```

Win	PM	Motif
Y	*Y*	*Y*

mixedData Sets the entry field to accept text that is a mixture of SBCS and DBCS characters. Conversion from an ASCII DBCS code page to an EBCDIC DBCS code page can result in a possible increase in the length of the data because of the addition of shift-in and shift-out characters, but it will not exceed the text limit of the entry field.

```
static const Style
  mixedData;
```

Win	PM	Motif
I	*Y*	*Y*

readOnly Prevents insertions or changes to the text in the entry field.

```
static const Style
  readOnly;
```

Win	PM	Motif
Y	*Y*	*Y*

rightAlign Sets the text in the entry field to be right-justified.

```
static const Style
  rightAlign;
```

Win	PM	Motif
Y	*Y*	*Y*

Win This style cannot be mixed with the autoScroll or unreadable styles.

sbcsData Sets the entry field to accept single-byte text only.

© IBM Corp. 1992, 1996

```
static const Style
  sbcsData;
```

Win	PM	Motif
I	*Y*	*Y*

unreadable Causes the text in the entry field to be displayed using an asterisk for each character.

```
static const Style
  unreadable;
```

Win	PM	Motif
Y	*Y*	*Y*

upperCase Displays the text in the entry field in uppercase characters.

```
static const Style
  upperCase;
```

Win	PM	Motif
Y	*N*	*N*

Inherited Public Data

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IEntryField contains the following nested classes:

IEntryField::Style (see page 297)

Alignment

```
Alignment {
    left,
    center,
    right
    };
```

Use these enumerators to specify the alignment of text in the entry field:

left
Left-justifies the text in the control.

center
Centers the text in the control.

right
Right-justifies the text in the control.

Motif X-Motif only supports the left alignment.

CharType

```
CharType {
    any,
    sbcs,
    dbcs,
    mixed
    };
```

Use these enumerators to specify the type of character data allowed in the entry field:

sbcs
Accepts single-byte (SBCS) text only.

dbcs
Accepts double-byte (DBCS) text only.

© IBM Corp. 1992, 1996

mixed

Accepts text that is a mixture of SBCS and DBCS characters. Conversion from an ASCII DBCS code page to an EBCDIC DBCS code page can result in an increase in the length of the text because of the addition of shift-in and shift-out characters. Using this enumerator prevents the overflow of EBCDIC string buffers.

any

Accepts text that is a mixture of SBCS and DBCS characters.

Note: If the text contains both single-byte and double-byte characters and is to be converted from an ASCII code page into an EBCDIC code page, this enumerator causes an entry field to not account for shift-in and shift-out characters that are introduced into its text. An overflow of EBCDIC string buffers might occur.

Motif X-Motif only supports the enumerator any.

IEntryField::Style

Derivation

IBase
  IBitFlag
    IEntryField::Style

Inherited By None.

Header File ientryfd.hpp

The nested class IEntryField::Style provides a set of valid styles for the IEntryField (p. 271) class.

Motif The AIX version does not support the centerAlign and rightAlign styles. The alignment is always leftAlign.

Motif does not support the command, dbcsData, sbcsData, and mixedData styles. The XmText widget does not support the unreadable style.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

© Copyright IBM Corp. 1992, 1996

IEntryField::Style

IBase		
recoverable	**unrecoverable**	

IEntryFieldNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          IEntryFieldNotifyHandler

Inherited By IComboBoxNotifyHandler

Header File ientrynh.hpp

Members

Member	Page
Constructor	299
dispatchHandlerEvent	300
~IEntryFieldNotifyHandler	299

The IEntryFieldNotifyHandler class processes events for all classes of entry fields.

This class is designed to handle events that require the entry field class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IEntryFieldNotifyHandler

This is the default constructor and accepts no parameters.

```
IEntryFieldNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IEntryFieldNotifyHandler

© Copyright IBM Corp. 1992, 1996

IEntryFieldNotifyHandler

```
virtual
  ~IEntryFieldNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the entry field observers if any of the following events are received:

- insertMode event
- dataUpdate event
- limit event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

ITextControlNotifyHandler		
dispatchHandlerEvent		

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IEnumHandle

Derivation

IBase
IHandle
IEnumHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	302
operator Value	302

The IEnumHandle class accesses lists of enumerated items. For example, an enumeration list can contain the handles of all the child windows for a given parent window.

PM IEnumHandle is an alias for the OS/2 Programmer's Toolkit type HENUM.

Motif AIX does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IEnumHandle Constructs objects of this class from an enumeration handle (a value of type IHandle::Value), which defaults to 0.

```
IEnumHandle( Value henum = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

© Copyright IBM Corp. 1992, 1996

```
operator Value() const;
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

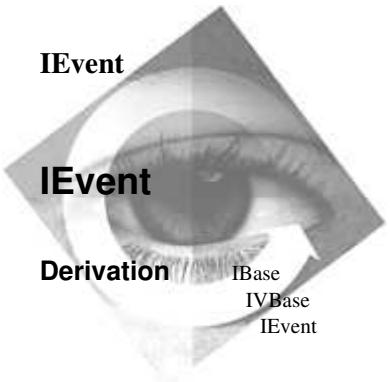
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IEvent

Derivation

IBase
IVBase
IEvent

Inherited By

ICnrDrawBackgroundEvent
ICommandEvent
IControlEvent
IDDEBeginEvent
IDDEEndEvent
IDDEEvent
IDMEvent
IFileDialogEvent
IFrameEvent
IHelpErrorEvent
IHelpHyperlinkEvent
IHelpMenuBarEvent
IHelpNotifyEvent
IHelpSubitemNotFoundEvent
IHelpTutorialEvent
IKeyboardEvent
IMenuEvent
IMMCuePointEvent
IMMDeviceEvent
IMMNotifyEvent
IMMPassDeviceEvent
IMMPositionChangeEvent
IMouseEvent
IMousePointerEvent
IPaintEvent
IResizeEvent
IScrollEvent

Header File ievent.hpp

Members

Member	Page	Member	Page
Constructor	306	passToOwner	309
controlHandle	309	result	309
controlWindow	310	setControlHandle	311
dispatchingWindow	310	setDispatchingHandle	311
eventId	307	setEventType	309
eventType	308	setHandle	312
handle	311	setPassToOwner	309
operator =	307	setResult	309
parameter1	308	window	312
parameter2	308	~IEvent	307

The IEvent class is the base-event information class. You can query event parameters and set event results. Additional functions are provided by the event classes derived from IEvent.

Motif This section describes how X-Motif events and their data are used to construct IEvent objects. This information is primarily for those who want to perform X-Motif-unique extensions or integrate custom widgets.

The basic Motif callback function signature is as follows:

© Copyright IBM Corp. 1992, 1996

```
void  callbackFn(
        Widget    widget,       /* widget that triggered callback */
        caddr_t   client_data,  /* client data                    */
        caddr_t   call_data);   /* data from the widget           */
```

On a callback, *call_data* is always a pointer to a structure. This structure contains at least the following fields:

```
   int     reason;         /* reason defined by Motif */
   XEvent* event           /* pointer to X event */
```

Depending on the reason and widget, additional fields may exist in the structure. You can use XmAnyCallbackStruct to refer to the preceding structure.

An Xt event registration callback function signature is as follows:

```
void  eventHandlerFn(
        Widget    widget,  /* widget in whose window event occurred*/
        caddr_t   client_data,  /* client data                    */
        XEvent   *event,        /* event data                     */
        ::Boolean continue_to_dispatch);
```

The XEvent structure is a union of different structures, one for each set of events with different attributes. The xany variant can be used to access the following common fields:

```
   int            type;          /* type of event                  */
   unsigned long  serial;        /* serial of last request processed*/
   Bool           send_event;    /* true if from client (XSendEvent)*/
   Display       *display;       /* display where occurred          */
   Window         window;        /* window where requested          */
```

See *O'Reilly Volume 6B* or any other book on X/Motif for further details.

The User Interface Class Library-supported event types (IEvent and derived classes) provide portable accessors to obtain event details, and they modify event data where appropriate. The responsibility for the portable access to event details generally lies with derived IEvent classes and handlers, which process the events. The implementations of these classes vary by event type and platform, but they use the consistent definition of the base IEvent structure. The library also provides nonportable accessors to access the raw event data. This facilitates both implementation of the library as well as extensions to it. The raw event data is presented in essentially the same way the underlying system represents it.

Public Functions

Constructors

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

© IBM Corp. 1992, 1996

IEvent

1 `IEvent( IWindow* window,`
`unsigned long eventId,`
`const IEventParameter1& parameter1,`
`const IEventParameter2& parameter2 );`

Win	PM	Motif
Y	Y	Y

Create an event using the specified IWindow (p. 1044) object, event ID, IEventParameter1 (p. 319) object, and IEventParameter2 (p. 320) object.

Motif In AIX, the IEvent constructor interprets its parameters as follows:

window
The widget parameter, as passed to the callback or event-handler function.

eventId
The "reason" from a Motif callback, or the type value from an Xt event registration. There is an exception to this in the IEvent generated by using sendEvent and postEvent.

parameter1
This parameter may contain either of the following:

- The *call_data* parameter from a Motif callback
- NULL, indicating an Xt event registration callback

Note: Because the *eventId*s from either type of callback might be the same value, you must use *parameter1* to distinguish whether the IEvents result from a Motif callback or from an Xt event registration.

parameter2
The *XEvent** parameter associated with the callback. For a Motif callback, the value is *call_data->event*. For an Xt event registration callback, the value is the *event* parameter on the callback routine as passed to the event-handler routine.

2 `IEvent( const IWindowHandle& handle,`
`unsigned long eventId,`
`const IEventParameter1& parameter1,`
`const IEventParameter2& parameter2 );`

Win	PM	Motif
Y	Y	Y

Create an event using the specified IWindowHandle (p. 1107) object, event ID, IEventParameter1 (p. 319) object, and IEventParameter2 (p. 320) object.

Motif In AIX, the IEvent constructor interprets its parameters as follows:

handle
The widget parameter, as passed to the callback or event-handler function.

eventId
The "reason" from a Motif callback, or the type value from an Xt event registration. There is an exception to this in the IEvent generated by using sendEvent and postEvent.

parameter1
This parameter may contain either of the following:

- The *call_data* parameter from a Motif callback
- NULL, indicating an Xt event registration callback

Note: Because the *eventId*s from either type of callback might be the same value, you must use *parameter1* to distinguish whether the IEvents result from a Motif callback or from an Xt event registration.

parameter2
The *XEvent** parameter associated with the callback. For a Motif callback, the value is *call_data->event*. For an Xt event registration callback, the value is the *event* parameter on the callback routine as passed to the event-handler routine.

3
```
IEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Create an event from another IEvent object.

operator = Assigns the value of one event object to another.

```
IEvent&
  operator =( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IEvent

```
virtual
  ~IEvent();
```

Win	PM	Motif
Y	Y	Y

Event Data

Use these members to set or query data for an event object, such as its event parameters and event result.

eventId Returns the ID for the event.

© IBM Corp. 1992, 1996

IEvent

```
unsigned long
  eventId() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif For Motif callback events, this function's return value is the callback reason. For Xt event-registration routines, this function's return value is the data member XEvent type, or the data member EventType for User Interface Class Library-defined ClientMessage events.

Note: Remember, if *parameter1* does not have a value of NULL, the event is a Motif callback event. If *parameter1* does have a value of NULL, the event is an Xt event registration routine.

eventType

Returns the type of the event. The event type indicates the kind of event that triggered the creation of the IEvent object. An IEvent object can be created in response to an XEvent, a Motif callback, or an event within the User Interface Class Library. The event type is determined and set by IWindow (p. 1044). Determine the event type before checking the IEvent::eventId (p. 307) because the identifiers may overlap for different event types.

```
EventType
  eventType() const;
```

Win	PM	Motif
N	*N*	*Y*

parameter1

Returns the first event parameter.

```
IEventParameter1
  parameter1() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The return value is either of the following:

- A pointer to the Motif callback structure
- A NULL, indicating an XEvent

parameter2

Returns the second event parameter.

```
IEventParameter2
  parameter2() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The return value is a pointer to the XEvent structure. The User Interface Class Library defines several ClientMessage events, which can appear in this structure. Specifically, the atom "ICLUI_MESSAGE" in the message_type field identifies these events. The User Interface Class Library handlers process the events.

passToOwner

IWindow::dispatch (p. 1082) calls this member function to determine if an IEvent object can be passed up the owner chain.

```
Boolean
  passToOwner() const;
```

Win	PM	Motif
N	N	Y

result

Returns the event result.

```
IEventResult
  result() const;
```

Win	PM	Motif
Y	Y	Y

setEventType

Set the type of the IEvent, which indicates the origin of the event that triggered the creation of the IEvent object. A User Interface Class Library user should not need to call this member function.

```
IEvent&
  setEventType( EventType typeOfEvent );
```

Win	PM	Motif
N	N	Y

setPassToOwner

Controls whether or not the IEvent object can propagate up the owner chain.

```
IEvent&
  setPassToOwner( Boolean passOn = true );
```

Win	PM	Motif
N	N	Y

setResult

Sets the event result.

1
```
IEvent&
  setResult( Boolean eventResult );
```

Win	PM	Motif
Y	Y	Y

2
```
IEvent&
  setResult( const IEventResult& eventResult );
```

Win	PM	Motif
Y	Y	Y

Event Window

Use these members to set or query the windows associated with an event object.

controlHandle

Returns the IWindowHandle (p. 1107) object for the presentation window that originated the event. If a valid object cannot be returned, 0 is returned.

© IBM Corp. 1992, 1996

Typically, this is a control, such as a button. For example, when an event has just been dispatched from an ICheckBox (p. 146) to an IHandler (p. 411), the return value is the same as that for dispatchingWindow (p. 310). If the event has been owner-routed and dispatched from the owner of the ICheckBox, the return value is different. controlWindow (p. 310) returns the ICheckBox, while dispatchingWindow returns the owner. This function provides consistent results for events of the following OS/2 types and their equivalents on other systems:

- WM_CONTROL
- WM_DRAWITEM
- WM_MEASUREITEM

```
IWindowHandle
  controlHandle() const;
```

Win	PM	Motif
Y	Y	Y

Motif On AIX, this function returns valid value for all event types.

controlWindow

Returns a pointer to the IWindow (p. 1044) object that originated the event. If a valid object cannot be returned, 0 is returned.

Typically, this is a control, such as a button. For example, when an event has just been dispatched from an ICheckBox (p. 146) to an IHandler (p. 411), the return value is the same as that for dispatchingWindow (p. 310). If the event has been owner-routed and dispatched from the owner of the ICheckBox, the return value is different. controlWindow (p. 310) returns the ICheckBox, while dispatchingWindow returns the owner. This function provides consistent results for events of the following OS/2 types and their equivalents on other systems:

- WM_CONTROL
- WM_DRAWITEM
- WM_MEASUREITEM

```
virtual IWindow*
  controlWindow() const;
```

Win	PM	Motif
Y	Y	Y

Motif On AIX, this function returns a valid value for all event types.

dispatchingWindow

Returns a pointer to the IWindow (p. 1044) object from which this event was just dispatched. The pointer can represent any of the following:

- The original object that originated the event
- The owner of that object if the event is owner-routed
- The owner's owner

This function gives consistent results across all systems so you can write portable code.

```
virtual IWindow*
  dispatchingWindow() const;
```

Win	PM	Motif
Y	Y	Y

handle

Returns an IWindowHandle (p. 1107) object for the IWindow (p. 1044) object the event has been dispatched to. Use of this function may result in nonportable code; use controlHandle (p. 309) instead.

```
IWindowHandle
  handle() const;
```

Win	PM	Motif
Y	Y	Y

setControlHandle

Sets the IWindowHandle (p. 1107) object for the presentation window that originated the event. This is initialized to the handle value passed in on the IEvent constructor. If a valid object cannot be set, 0 should be set.

Typically, this is a control, such as a button. For example, when an event has just been dispatched from an ICheckBox (p. 146) to an IHandler (p. 411), this value should be the same as that for dispatchingWindow (p. 310). If the event has been owner-routed and dispatched from the owner of the ICheckBox, this value should be the ICheckBox, while dispatchingWindow returns the owner.

```
IEvent&
  setControlHandle( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

setDispatchingHandle

Sets the IWindowHandle (p. 1107) object for the IWindow (p. 1044) object from which this event was just dispatched. This value is set by IWindow during the event-dispatching process. The value can represent any of the following:

- The original object that originated the event
- The owner of that object if the event is owner-routed
- The owner's owner

This function gives consistent results across all systems so you can write portable code.

```
IEvent&
  setDispatchingHandle( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

setHandle

Sets the IWindowHandle (p. 1107) object for the IWindow (p. 1044) object the event has been dispatched to. This value is set by IWindow during the event-dispatching process. Use of this function may result in nonportable code; use setControlHandle (p. 311) or setDispatchingHandle (p. 311) instead.

```
IEvent&
  setHandle( const IWindowHandle& handle );
```

Win	PM	Motif
Y	*Y*	*Y*

window

Returns a pointer to the IWindow (p. 1044) object the event has been dispatched to. Use of this function may result in nonportable code; use controlWindow (p. 310) or dispatchingWindow (p. 310) instead.

```
virtual IWindow*
  window() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

EventType

```
EventType {
    Xevent,
    MotifCallback,
    Library
    };
```

The preceding enumerators specify the origin of the event that triggered the creation of the IEvent object.

IEventData

Derivation

IBase
  IEventData

Inherited By

None.

Header File

ievtdata.hpp

Members

Member	Page	Member	Page
Constructor	314	highNumber	316
asLong	317	lowHighByte	316
asUnsignedLong	317	lowLowByte	316
char1	315	lowNumber	316
char2	315	number1	316
char3	315	number2	316
char4	315	operator char *	317
highHighByte	315	operator unsigned long	317
highLowByte	316		

The IEventData class encapsulates the data of an event. IEventData is interchangeable with the following classes, which are actually aliases (that is, typedefs) for IEventData:

- IEventParameter1 (p. 319)
- IEventParameter2 (p. 320)
- IEventResult (p. 321)
- IHighEventParameter (p. 465)
- ILowEventParameter (p. 524)

Motif Although the User Interface Class Library sometimes internally uses the IEventResult value returned by the application, this value is not returned to the X-Motif window system. X-Motif does not support returning any value. Particularly, it does not support the ability to prevent further processing of the event by others, as Presentation Manager does.

The interpretation of the IEventData objects in an IEvent are system-dependent. See IEvent (p. 304) for a description of how the fields are interpreted in each environment. For applications you want to be portable, you should use accessor functions provided by derived classes of IEvent to access event data.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

You can construct and destruct objects of this class.

IEventData

1 `IEventData();`

Win	PM	Motif
Y	Y	Y

Create an IEventData object with the event data set to 0.

2 `IEventData( void* value );`

Win	PM	Motif
Y	Y	Y

Create an IEventData object from a pointer to a void. The event data is set to the specified void pointer.

3 `IEventData( unsigned long value );`

Win	PM	Motif
Y	Y	Y

Create an IEventData object from an unsigned long integer value. The event data is set to the specified unsigned long integer.

4 `IEventData( int value );`

Win	PM	Motif
Y	Y	Y

Create an IEventData object from an integer. The event data is set to the specified integer. This constructor accepts an uncasted value of 0.

5 `IEventData( BooleanConstants value );`

Win	PM	Motif
Y	Y	Y

Create an IEventData object from a BooleanConstants value. This constructor accepts one of the two IBase::BooleanConstants values, true or false. The event data is set to the Boolean value.

6
```
IEventData( unsigned short lowValue,
            unsigned short hiValue );
```

Win	PM	Motif
Y	Y	Y

Create an IEventData object from two unsigned short integer values. The event data is set to an unsigned long integer whose two words are the two specified unsigned short integers.

7
```
IEventData( unsigned short lowValue,
            char lowByte,
            char hiByte );
```

Win	PM	Motif
Y	Y	Y

Create an IEventData object using the specified unsigned short value and two characters. The event data is set to an unsigned long integer whose two words are the unsigned short integer in the low word and the two characters in the high word.

Contents

These members query and set the event data contained by objects of this class.

char1

Returns the character in the low byte of the event data's low word. This is the same as lowLowByte (p. 316).

```
char
  char1() const;
```

Win	PM	Motif
Y	Y	Y

char2

Returns the character in the high byte of the event data's low word. This is the same as lowHighByte (p. 316).

```
char
  char2() const;
```

Win	PM	Motif
Y	Y	Y

char3

Returns the character in the low byte of the event data's high word. This is the same as highLowByte (p. 316).

```
char
  char3() const;
```

Win	PM	Motif
Y	Y	Y

char4

Returns the character in the high byte of the event data's high word. This is the same as highHighByte (p. 315).

```
char
  char4() const;
```

Win	PM	Motif
Y	Y	Y

highHighByte

Returns the character in the high byte of the event data's high word. This is the same as char4 (p. 315).

© IBM Corp. 1992, 1996

IEventData

```
char
  highHighByte() const;
```

Win	PM	Motif
Y	Y	Y

highLowByte Returns the character in the low byte of the event data's high word. This is the same as char3 (p. 315).

```
char
  highLowByte() const;
```

Win	PM	Motif
Y	Y	Y

highNumber Returns the event data's high word. This is the same as number2 (p. 316).

```
unsigned short
  highNumber() const;
```

Win	PM	Motif
Y	Y	Y

lowHighByte Returns the character in the high byte of the event data's low word. This is the same as char2 (p. 315).

```
char
  lowHighByte() const;
```

Win	PM	Motif
Y	Y	Y

lowLowByte Returns the character in the low byte of the event data's low word. This is the same as char1 (p. 315).

```
char
  lowLowByte() const;
```

Win	PM	Motif
Y	Y	Y

lowNumber Returns the event data's low word. This is the same as number1 (p. 316).

```
unsigned short
  lowNumber() const;
```

Win	PM	Motif
Y	Y	Y

number1 Returns the event data's low word. This is the same as lowNumber (p. 316).

```
unsigned short
  number1() const;
```

Win	PM	Motif
Y	Y	Y

number2 Returns the event data's high word. This is the same as highNumber (p. 316).

```
unsigned short
  number2() const;
```

Win	PM	Motif
Y	Y	Y

Conversion

These members convert an IEventData object to a four-byte value. You can either explicitly convert the object by calling a member function or casting, or allow the compiler to implicitly convert the object using an operator.

asLong Returns the long event data value.

```
long
  asLong() const;
```

Win	PM	Motif
Y	Y	Y

asUnsignedLong

Returns the unsigned long event data value.

```
unsigned long
  asUnsignedLong() const;
```

Win	PM	Motif
Y	Y	Y

operator char *

Returns the event data as a pointer to a character.

```
operator char *() const;
```

Win	PM	Motif
Y	Y	Y

operator unsigned long

Returns the event data as an unsigned long number.

```
operator unsigned long() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

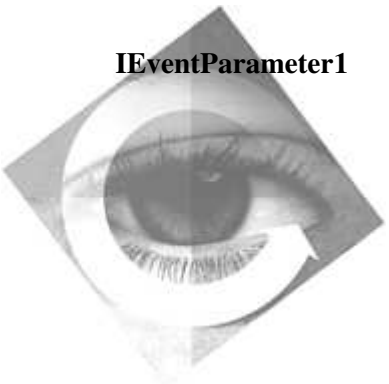
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

© IBM Corp. 1992, 1996

IEventData

IBase		
recoverable	**unrecoverable**	



IEventParameter1

Derivation IBase
IEventParameter1

Inherited By None.

Header File ievtdat2.hpp

The IEventParameter1 class encapsulates the message parameter (MPARAM) and result (MRESULT) data of an event.

IEventParameter1 is an alias of the IEventData class. See IEventData (p. 313) for more information.

In addition, IEventParameter1 is interchangeable with the following classes:

IEventParameter2 (p. 320)
IEventResult (p. 321)
IHighEventParameter (p. 465)
ILowEventParameter (p. 524)

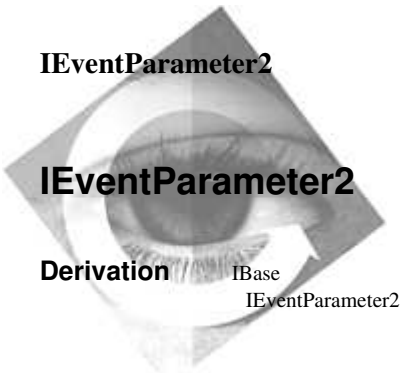
Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



IEventParameter2

Derivation IBase
IEventParameter2

Inherited By None.

Header File ievtdat2.hpp

The IEventParameter2 class encapsulates the message parameter (MPARAM) and result (MRESULT) data of an event.

IEventParameter2 is an alias of the IEventData class. See IEventData (p. 313) for more information.

In addition, IEventParameter2 is interchangeable with the following classes:

IEventParameter1 (p. 319)
IEventResult (p. 321)
IHighEventParameter (p. 465)
ILowEventParameter (p. 524)

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IEventResult

Derivation IBase
IEventResult

Inherited By None.

Header File ievtdat2.hpp

The IEventResult class encapsulates the message parameter (MPARAM) and message result (MRESULT) data of an event.

IEventResult is an alias of the IEventData class. See IEventData (p. 313) for more information.

In addition, IEventResult is interchangeable with the following classes:

IEventParameter1 (p. 319)
IEventParameter2 (p. 320)
IHighEventParameter (p. 465)
ILowEventParameter (p. 524)

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IFocusHandler

Derivation

IBase
IVBase
IHandler
IFocusHandler

Inherited By None.

Header File ifocshdr.hpp

Members

Member	Page	Member	Page
Constructor	323	lostFocus	324
dispatchHandlerEvent	324	~IFocusHandler	323
gotFocus	324		

The IFocusHandler class handles events occurring when a control gains or loses the input focus, such as when the user tabs to or from an entry field. These objects handle the processing of focus change events for the following controls:

IContainerControl (Vol. III)
IEntryField (p. 271)
IListBox (p. 495)
IMultiLineEdit (p. 641)
ISlider (p. 874)
ICircularSlider (Vol. III)
INumericSpinButton (p. 672)
ITextSpinButton (p. 963)

Create a handler derived from IFocusHandler and attach it to the control that gets or loses the input focus, or to the owner window of the control. You can do this by calling IHandler::handleEventsFor (p. 413) to pass this control or owner window to the focus handler.

When the focus handler receives a focus event, it creates an object of IControlEvent (p. 227) and routes that object to the appropriate IFocusHandler virtual function. You can override these virtual functions to supply your own specialized processing of a focus event.

The return value from the virtual functions specifies whether the focus event is passed on for additional processing, as follows:

© Copyright IBM Corp. 1992, 1996

true The focus event requires no additional processing. Do not pass it to another handler.

false Pass the focus event to the next handler for additional processing, as follows:

- If there is another handler for the control, pass the focus event to the next handler.
- If this is the last handler for the control, call IWindow::dispatch (p. 1082) to dispatch the focus event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the focus event.

In Motif, IFocusHandlers cannot handle the processing of focus change events for ISlider (p. 874) controls.

Public Functions

Constructors

You can construct and destruct objects of this class.

IFocusHandler

Constructs the default focus handler.

```
IFocusHandler();
```

Win	PM	Motif
Y	Y	Y

~IFocusHandler

```
virtual
  ~IFocusHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

© IBM Corp. 1992, 1996

IFocusHandler

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IFocusHandler object processes only focus-change events.

dispatchHandlerEvent

If a focus event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

A focus handler allows you to process focus changes through its event-processing members. Override at least one of these virtual functions in a derived class.

gotFocus

Implemented by a derived class to process a gain-focus event.

```
virtual Boolean
  gotFocus( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

lostFocus

Implemented by a derived class to process a lost-focus event.

Note: It is possible that a set-focus event can be ahead of a related lost-focus event on the message queue.

```
virtual Boolean
  lostFocus( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

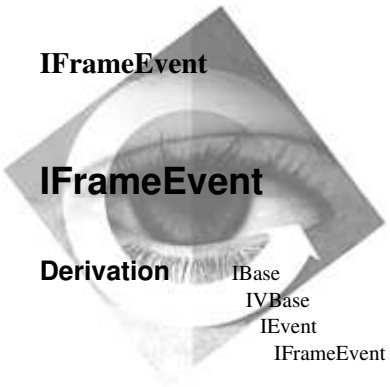
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IFrameEvent

Derivation

IBase
IVBase
IEvent
IFrameEvent

Inherited By IFrameFormatEvent

Header File iframevt.hpp

Members

Member	Page	Member	Page
Constructor	326	operator =	327
frame	327	~IFrameEvent	327

The frame window handler creates objects of the IFrameEvent class and of classes derived from IFrameEvent. The frame window handler dispatches these objects to the handler's virtual function specific to a frame event.

Mainly, this class provides a common means of obtaining the frame window associated with the event.

Public Functions

Constructors

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

IFrameEvent

1 `IFrameEvent( const IFrameEvent& frameEvent );`

Win	PM	Motif
Y	*Y*	*Y*

Construct an IFrameEvent by copying the specified event.

2 `IFrameEvent( const IEvent& baseEvent );`

Win	PM	Motif
Y	*Y*	*Y*

Construct an IFrameEvent using the specified event.
IFrameHandler::dispatchHandlerEvent (p. 347) constructs objects of this class from an

© Copyright IBM Corp. 1992, 1996

object of the class IEvent (p. 304). The event's window handle must be that of a frame window.

operator = Replaces the event with the specified one.

```
IFrameEvent&
  operator =( const IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	Y

~IFrameEvent

```
~IFrameEvent();
```

Win	PM	Motif
Y	Y	Y

Frame Window

Use frame window members to query the frame window associated with this event.

frame Returns a pointer to the frame window receiving this event.

```
IFrameWindow*
  frame() const;
```

Win	PM	Motif
Y	Y	Y

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	

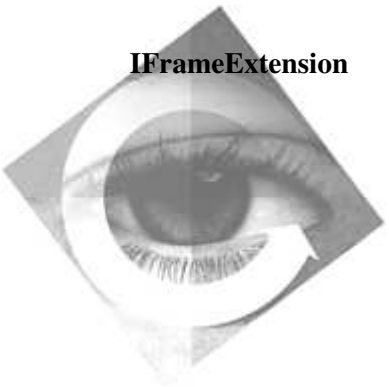
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

IFrameEvent

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFrameExtension

Derivation

IBase
  IFrameExtension

Inherited By

None.

Header File

iframext.hpp

Members

Member	Page	Member	Page
Constructor	330	separatorType	336
attachedToId	332	separatorWidth	336
attachTo	332	setSeparatorHandle	336
baseRectFor	333	setSize	333
control	335	sizeTo	334
drawSeparator	335	totalRectFor	334
fixedSize	333	type	335
location	332	useMinimumSize	334
relativeSize	333	~IFrameExtension	331
separatorHandle	336		

IFrameWindow (p. 349) maintains objects of the IFrameExtension class to record information about each of its frame-extension windows. You add a frame extension to a frame window by calling IFrameWindow::addExtension (p. 361). You do not need to create an IFrameExtension object yourself.

IFrameWindow::extensions (p. 378) and IFrameWindow::setExtensions (p. 379) deal with collections of such frame extensions. Each collection is actually an object of the IFrameExtensions (p. 338) class, which is essentially a synonym for ISequence<IFrameExtension*>. The class ISequence is in the Collection Class library.

Frame extensions have the following attributes:

type Specifies the extension be one of three types having the following characteristics:

- A fixed width or height.
- A width or height that is a fraction of the frame control it occupies. See the *location* attribute.
- A width or height determined by the minimum size of the extension window.

© Copyright IBM Corp. 1992, 1996

size Specifies that the size is any of the following:

- A specific width or height, for fixed-size extensions
- A fraction, for relative-sized extensions
- The minimum size of the associated control.

location Specifies the portion of the frame the extension occupies. The enumeration IFrameWindow::Location (p. 389) provides the possible values.

control Specifies a pointer to the control window that occupies the extension.

Public Functions

Constructors

Use these functions to construct objects of the IFrameExtension class. You cannot copy or assign IFrameExtension objects because both the copy constructor and assignment operator are private functions. You can construct objects of this class by providing the same set of parameters as for IFrameWindow::addExtension (p. 361).

IFrameExtension

1
```
IFrameExtension( IWindow* window,
                 IFrameWindow::Location location,
                 unsigned long widthOrHeight,
                 IFrameWindow::SeparatorType separator );
```

Win	PM	Motif
Y	Y	Y

Use this constructor for a fixed-type extension.

You construct objects of this class by providing the following set of parameters.

window The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, client area) to be occupied by the frame extension. Use the enumeration IFrameWindow::Location (p. 389) to specify the location for the frame extension.

widthOrHeight
The portion of the frame window area allocated to the frame extension, in a fixed width or height.

separator The separator type. Use the enumeration IFrameWindow::SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

2 `IFrameExtension( IWindow* window, IFrameWindow::Location location, double percentage, IFrameWindow::SeparatorType separator );`

Win	PM	Motif
Y	Y	Y

Create relative-type extensions using this constructor.

You can construct objects of this class by providing the following set of parameters:

window The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, client area) to be occupied by the frame extension. Use the enumeration IFrameWindow::Location (p. 389) to specify the location for the frame extension.

percentage The portion of the frame window area allocated to the frame extension, in a floating point percentage (0 < x < 1.0).

separator The separator type. Use the enumeration IFrameWindow::SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

3 `IFrameExtension( IWindow* window, IFrameWindow::Location location, IFrameWindow::SeparatorType separator );`

Win	PM	Motif
Y	Y	N

Use this constructor to create frame extensions that are sized to the minimum size of the associated window.

You can construct objects of this class by providing the same set of parameters as for IFrameWindow::addExtension (p. 361).

window The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, client area) to be occupied by the frame extension. Use the enumeration IFrameWindow::Location (p. 389) to specify the location for the frame extension.

separator The separator type. Use the enumeration IFrameWindow::SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

~IFrameExtension

```
virtual
  ~IFrameExtension();
```

Win	PM	Motif
N	N	Y

© IBM Corp. 1992, 1996

Extension Location

When adding a frame extension, you specify where the frame extension is attached. See IFrameWindow::Location (p. 389) for details on locations. You can attach the frame extension relative to the client area, the menu bar, or the title bar. You can query the location of the extension as well as the ID of the frame control to which the extension is attached.

attachedToId Returns the ID of the frame control this extension is attached to.

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

Win	PM	Motif
Y	Y	Y

location Returns the frame extension's location.

```
IFrameWindow::Location
  location() const;
```

Win	PM	Motif
Y	Y	Y

Extension Sizing

Using the frame-extension sizing members, you can query and set the size of the frame extension.

You can query the width and height of relative- and fixed-type extensions. See the Extension Type (p. 335) group for information on types of extensions.

You can reset the size of an extension or indicate that the extension is to be sized to the minimum size of the window it contains. The size of a separator, if requested, is not included in the size of the extension.

Various sizing calculations are provided. You can calculate a total rectangle based on a specified rectangle plus the frame extension's size. This total also includes the separator width. The opposite calculation function is also provided. You specify a base rectangle and the rectangle size is returned minus the frame extension's size and separator width.

attachTo Returns the SWP structure which contains information about how the frame extension is sized when attached to the control. The SWP structure is updated to account for the space this extension occupies.

```
virtual ISWP
  attachTo( ISWP& baseSWP );
```

Win	PM	Motif
Y	Y	I

baseSWP The position of the control that the frame extension is attached to.

baseRectFor

Calculates and returns a rectangle representing the client rectangle without its frame extension. Both the rectangle you specify and the returned rectangle are in the orientation of the native windowing system. You can use the class ICoordinateSystem (p. 230) to determine the native orientation.

```
virtual IRectangle
  baseRectFor( const IRectangle& totalRect ) const;
```

Win	PM	Motif
Y	Y	Y

totalRect The total rectangle, which is the client rectangle plus the frame extension's size and position.

Motif This function always returns a default IRectangle (Vol. I).

fixedSize

Returns the fixed width or height in window coordinates. If the extension is of type relative, 0 is returned.

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

Win	PM	Motif
Y	Y	Y

relativeSize

Returns the relative width or height as a percentage. If the extension is of type fixed, 0.0 is returned.

```
virtual double
  relativeSize() const;
```

Win	PM	Motif
Y	Y	Y

setSize

Resets the size of the extension.

1
```
virtual IFrameExtension&
  setSize( double widthOrHeight );
```

Win	PM	Motif
Y	Y	Y

Use this setSize function to set the size of a relative-type extension.

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a floating point percentage (double, 0 < x < 1.0).

2
```
virtual IFrameExtension&
  setSize( unsigned long widthOrHeight );
```

Win	PM	Motif
Y	Y	Y

Use this setSize function to set the size of a fixed-type extension.

© IBM Corp. 1992, 1996

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a fixed width or height (unsigned long).

3
```
virtual IFrameExtension&
  setSize( int widthOrHeight );
```

Win	PM	Motif
Y	Y	N

Use this setSize function to set the size of a fixed-type extension.

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a fixed width or height (int).

sizeTo Calculates and returns the size this extension would be if attached to a control of the specified size.

```
virtual ISize
  sizeTo( const ISize& baseSize ) const;
```

Win	PM	Motif
Y	Y	Y

baseSize The size of the proposed control.

Motif AIX does not support this function. The function always returns the specified *baseSize* value.

totalRectFor Calculates and returns a new rectangle value. This value is *baseRect* plus the frame extension's size and position. Both the rectangle you specify and the returned rectangle are in the orientation of the native windowing system. You can use the class ICoordinateSystem (p. 230) to determine the native orientation.

```
virtual IRectangle
  totalRectFor( const IRectangle& baseRect ) const;
```

Win	PM	Motif
Y	Y	Y

baseRect A rectangle representing the client rectangle without its frame extension.

Motif This function always returns a default IRectangle (Vol. I).

useMinimumSize
Specifies that the width or height of the extension is assigned from the minimum size of the window it contains.

```
virtual IFrameExtension&
  useMinimumSize();
```

Win	PM	Motif
Y	Y	N

Extension Type

When adding a frame extension, you specify the type of extension using one of the Type (p. 337) enumerations: fixed, relative, or minimumSize.

A fixed extension remains a specific size.

When adding a relative extension, you specify the percentage of the frame control that the extension will occupy. The extension is sized when the frame control is resized.

You can specify that the extension be sized according to its associated control's minimum size.

The type of extension you choose depends on the desired layout of your controls.

type Returns the extension type for this frame extension object. The returned value is an enumerator provided by Type (p. 337).

```
Type
  type() const;
```

Win	PM	Motif
Y	Y	Y

Extension Window

Use these members to query the window that is associated with the extension. This control is the control specified when the extension was added to the frame.

control Returns a pointer to the window that occupies this frame extension.

```
IWindow*
  control() const;
```

Win	PM	Motif
Y	Y	Y

Separator

A *separator* is a visual delineator between an extension and the window to which it is attached. You specify whether you want a separator when adding a frame extension by using IFrameWindow::addExtension (p. 361). You can specify no separator, a thin separator line, or a thick separator line. The separator types are defined in the enumeration IFrameWindow::SeparatorType (p. 390). You can query the width of the separator line. The separator width is not included in the size of the frame extensions. See the Extension Sizing (p. 332) members for more details.

drawSeparator

Draws the separator line between a frame extension and the adjacent control (or extension) it is attached to.

© IBM Corp. 1992, 1996

IFrameExtension

```
virtual void
  drawSeparator( const IPresSpaceHandle& psh );
```

Win	PM	Motif
Y	Y	I

separatorHandle

Returns the handle of the separator.

```
virtual IWindowHandle
  separatorHandle() const;
```

Win	PM	Motif
N	N	Y

separatorType

Returns the separator type for this frame extension object. The returned value is an enumerator provided by IFrameWindow::SeparatorType (p. 390).

```
virtual IFrameWindow::SeparatorType
  separatorType() const;
```

Win	PM	Motif
N	N	Y

separatorWidth

Returns the width of the separator line between this frame extension and either of the following:

- The control it is attached to
- The extension it is adjacent to

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

Win	PM	Motif
Y	Y	Y

setSeparatorHandle

Sets the handle of the separator object.

```
virtual IFrameExtension&
  setSeparatorHandle( const IWindowHandle& separatorHandle );
```

Win	PM	Motif
N	N	Y

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Type

```
Type {
    fixed,
    relative,
    minimumSize
    };
```

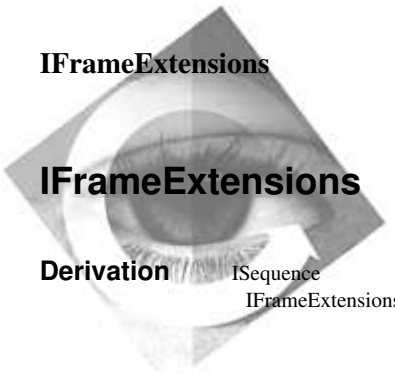
Use these enumerators to specify how the frame extension is sized:

fixed
Specifies a fixed width or height.

relative
Specifies a relative width or height based on how much of the attached-to frame control the extension occupies.

minimumSize
Specifies that the extension should be sized according to the minimum size of the frame extension's window.

© IBM Corp. 1992, 1996



IFrameExtensions

Derivation ISequence
IFrameExtensions

Inherited By None.

Header File iframext.hpp

Members

Member	Page
Constructor	338
~IFrameExtensions	338

The IFrameExtensions class is a wrapper for a set, or sequence, of IFrameExtension (p. 329) objects. The User Interface Class Library uses the IFrameExtensions class to add a level of indirection in the IFrameWindow (p. 349) interface.

The library derives the IFrameExtensions class from the class ISequence, which is in the Collection Class library.

Public Functions

Constructor

You can construct or destruct objects of this class. IFrameWindow (p. 349) objects use this class to maintain their frame extensions.

IFrameExtensions

Provides the default constructor.

```
IFrameExtensions();
```

Win	PM	Motif
Y	Y	Y

~IFrameExtensions

```
~IFrameExtensions();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

IFrameFormatEvent

Derivation

IBase
  IVBase
    IEvent
      IFrameEvent
        IFrameFormatEvent

Inherited By None.

Header File iframevt.hpp

Members

Member	Page	Member	Page
Constructor	339	swpArray	340
clientRect	340	~IFrameFormatEvent	340
setClientRect	340		

The IFrameFormatEvent class represents information about a frame format event. The frame window handler creates and dispatches objects of this class to IFrameHandler::format (p. 346) when a frame format event occurs.

Public Functions

Constructors

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

IFrameFormatEvent

1 `IFrameFormatEvent( const IEvent& baseEvent );`

Win	PM	Motif
Y	*Y*	*I*

Construct an IFrameFormatEvent using the specified event. IFrameHandler::dispatchHandlerEvent (p. 347) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to IFrameHandler::format (p. 346). The event's window handle must be that of a frame window.

PM The event object is expected to contain the following:

- WM_FORMATFRAME for the event ID.

© Copyright IBM Corp. 1992, 1996

- The event's first parameter is a pointer to an array of SWP (p. 934) structures. These dimensions of the array must be equal to the frame control count returned on the preceding WM_QUERYFRAMECTLCOUNT message.
- The event's second parameter is a pointer to a RECTL structure in which the client control's size and position are returned.

The event result field will ultimately contain the count of the number of frame controls processed.

2 `IFrameFormatEvent( const IFrameFormatEvent& fmtEvent );`

Win	PM	Motif
Y	Y	I

Construct an IFrameFormatEvent by copying the specified event.

~IFrameFormatEvent

```
~IFrameFormatEvent();
```

Win	PM	Motif
Y	Y	I

Formatting Information

Use these functions to query and set the position and size of frame extensions and other child windows of a frame window. You can query the ISWP (p. 934) objects associated with the event. You can query and set the client rectangle associated with this frame event using members from this group.

clientRect Returns the client rectangle associated with this event. The rectangle is always in the orientation of the native windowing system. You can use the class ICoordinateSystem (p. 230) to determine the native orientation.

```
IRectangle
  clientRect() const;
```

Win	PM	Motif
Y	Y	I

setClientRect Sets the client rectangle. You specify the rectangle in the orientation of the native windowing system. You can use the class ICoordinateSystem (p. 230) to determine the native orientation.

```
IFrameFormatEvent&
  setClientRect( const IRectangle& rectangle );
```

Win	PM	Motif
Y	Y	I

aRect The client rectangle value.

swpArray Returns a pointer to the ISWP (p. 934) array associated with this event.

```
ISWP*
  swpArray() const;
```

Win	PM	Motif
Y	*Y*	*I*

Inherited Public Functions

IFrameEvent		
frame	operator =	

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

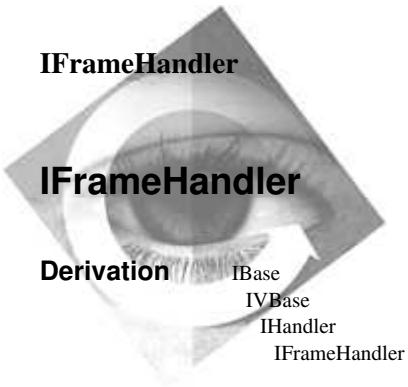
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IFrameHandler

Derivation

IBase
IVBase
IHandler
IFrameHandler

Inherited By None.

Header File iframhdr.hpp

Members

Member	Page	Member	Page
Constructor	344	draw	345
activated	344	format	346
calcRect	344	numberOfExtensions	346
closed	345	positionExtensions	348
command	345	saved	346
deactivated	345	~IFrameHandler	344
dispatchHandlerEvent	347		

The IFrameHandler class handles frame window events. Usually, when you construct an IFrameWindow (p. 349) object, the function IFrameWindow::start (p. 365) attaches an IFrameHandler object to the frame window.

When you use any of the protected member functions, IFrameWindow::create (p. 379), IFrameWindow::tryToLoadDialog (p. 382), or IFrameWindow::initialize (p. 381), and you do not specify the style IFrameWindow::deferCreation (p. 383), the frame constructor calls IFrameWindow::start.

If you construct the frame window using some alternate means (for example, by a derived class of IFrameWindow) and IFrameWindow::start is not called, either call IFrameWindow::addDefaultHandler (p. 379) to cause the frame to attach an IFrameHandler object to itself or create a handler derived from IFrameHandler and attach it to the IFrameWindow object. You can attach the handler derived from IFrameHandler by calling IHandler::handleEventsFor (p. 413) to pass the appropriate frame window to the handler.

To attach a handler derived from IFrameHandler to a frame window that already has an IFrameHandler object attached to it, you must first call IFrameWindow::removeDefaultHandler (p. 382). Otherwise, events can be processed twice, causing unpredictable results.

© Copyright IBM Corp. 1992, 1996

IFrameHandler dispatches the following events:

- Events that activate, close, deactivate, draw, and save the frame window.
- Command events (p. 210).
- Events used to size and position frame extensions. IFrameFormatEvent (p. 339) discusses this in more detail.

When the frame handler receives one of these frame events, it creates a corresponding event object and routes that object to the appropriate IFrameHandler virtual function. You can override these virtual functions to supply your own specialized processing of a frame event.

The return value from the virtual functions specifies whether the frame event is passed on for additional processing, as follows:

true The frame event requires no additional processing. Do not pass it to another handler.

false Pass the frame event to the next handler for additional processing, as follows:

- If there is another handler for the frame window, pass the frame event to the next handler.
- If this is the last handler for the frame window, call IWindow::defaultProcedure (p. 1082) to process the frame event.

Note: Because IFrameHandler processes ICommandEvent (p. 210) objects, think of it as if you had more than one ICommandHandler (p. 214) attached to a window. If you attach an IFrameHandler to the same window as an ICommandHandler, your ICommandHandler might not be called, depending on the order in which you attach these handlers.

The AIX environment does not support the following:

IFrameHandler::calcRect (p. 344)
IFrameHandler::deactivated (p. 345)
IFrameHandler::format (p. 346)

Public Functions

Constructors

You can construct and destruct objects of this class.

© IBM Corp. 1992, 1996

IFrameHandler

The default constructor accepts no parameters.

```
IFrameHandler();
```

Win	PM	Motif
Y	Y	Y

~IFrameHandler

```
virtual
  ~IFrameHandler();
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members handle frame-specific window events. Derived classes can override any of these virtual functions to provide specific functionality.

activated Called when the frame window is activated. By default, this function does nothing except return false. Derived classes can implement this function as follows:

- Performing their event processing
- Returning false so other handlers can act on this event

```
virtual Boolean
  activated( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	Y

Motif This function is called when the WM_TAKE_FOCUS message is received from the Motif Window Manager.

calcRect Calculates the size of either of the following:

- The client rectangle given a frame rectangle
- The frame rectangle given a client rectangle

This function accounts for the size needed for the frame's extensions and other standard components used by the frame, such as a title bar, menu bar, and system menu.

```
virtual Boolean
  calcRect( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function. The function always returns false.

closed

Called to close the frame window. This function does the following:

1. Queries whether the frame is modal.
2. If so, it dismisses the frame.
3. If the frame has a client window, the handler sends an SC_CLOSE system command to the client.
4. Calls the frame's IFrameWindow::willDestroyOnClose (p. 374) function.
5. If IFrameWindow::willDestroyOnClose returns true, the handler destroys the window.

The IFrameHandler::closed function always returns true.

Note: If the frame window has a client window, you can locate the processing for closing the window in a command handler attached to either the client or frame window rather than having to override the IFrameHandler::closed function. The command handler, however, does not allow you to prevent a frame window from closing.

```
virtual Boolean
  closed( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The system failed to destroy the specified window. A possible cause is an invalid window handle.

command

Called to process an application command event. If the frame has a client window, this function sends the event to the client window. This function always returns true.

```
virtual Boolean
  command( ICommandEvent& cmdEvent );
```

Win	PM	Motif
Y	Y	Y

deactivated

Called when the frame window loses the input focus. By default, this function does not do anything except return false. Derived classes can implement this function as follows:

- Performing their event processing
- Returning false so other handlers can act on this event

```
virtual Boolean
  deactivated( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	I

draw

Called to draw the frame window. This function draws the separator lines between frame extensions and returns false. Derived classes can implement their own version of this function as follows:

© IBM Corp. 1992, 1996

- Performing any additional drawing they require
- Returning false so other handlers can act on this event

```
virtual Boolean
  draw( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	I

format Fills in an array of window position structures, one for each frame control. Use this function to do this for the frame extensions added to the frame using IFrameWindow::addExtension (p. 361). Derived classes can implement this function as follows:

1. Calling IFrameHandler::format (p. 346)
2. Formatting their controls
3. Returning true, so the standard frame controls are not formatted again

```
virtual Boolean
  format( IFrameFormatEvent& formatEvent );
```

Win	PM	Motif
Y	Y	I

saved Called to save the application status. By default, this function does not do anything except return false. Derived classes can implement this function as follows:

- Saving the application status
- Returning false so other handlers can act on this event

```
virtual Boolean
  saved( IFrameEvent& frameEvent );
```

Win	PM	Motif
Y	Y	Y

Frame Extensions

Use these members to manage the frame extensions belonging to a frame window.

numberOfExtensions

Returns the number of frame extensions the specified frame window has. This number is also the number of elements in the collection returned by the frame's IFrameWindow::extensions (p. 378) function.

```
virtual unsigned long
  numberOfExtensions( const IFrameWindow* frame ) const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IFrameHandler object processes only frame-window-related events.

dispatchHandlerEvent

If a frame event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Motif For AIX, note the following frame events (or messages):

WM_DELETEWINDOW	Dispatches the closed function.
WM_SAVE_YOURSELF	Dispatches the saved function.
WM_TAKE_FOCUS	Dispatches the activated function.
WM_CLOSE	Dispatches the closed function.
SC_CLOSE	Dispatches the closed function.
WM_COMMAND	Dispatches the command function.

Frame Extensions

Use these members to manage the frame extensions belonging to a frame window.

© IBM Corp. 1992, 1996

positionExtensions

Positions the frame extensions, based on the size and position of standard frame components not added through IFrameExtension (p. 329) objects, such as the client window, title bar, menu bar, and system menu. Call this function from within the format (p. 346) function after you have modified the ISWP (p. 934) objects for standard frame controls. Return true from format to prevent the default frame handler from positioning the frame extensions again.

```
virtual unsigned long
  positionExtensions( IFrameFormatEvent& formatEvent,
                      unsigned long numStdControls );
```

Win	PM	Motif
Y	*Y*	*I*

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IFrameWindow

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow

Inherited By

IFileDialog
IFontDialog
IToolBarFrameWindow

Header File

iframe.hpp

Members

Member	Page	Member	Page
Constructor	357	enableNotification	369
accelerator	384	endFlashing	375
activateId	384	extensions	378
addDefaultHandler	379	findExtension	379
addExtension	361	findExtensions	377
addToWindowList	376	frameRectFor	368
alignNoAdjust	384	handleFor	370
animated	385	hideButton	386
appDBCSStatus	385	horizontalScroll	386
attachClient	377	icon	352
backgroundColor	356	initialize	381
beginFlashing	375	isAnExtension	363
border	385	isFlashing	373
borderHeight	353	isFrameWindow	377
borderSize	353	isMaximized	373
borderWidth	353	isMinimized	373
classDefaultStyle	385	isModal	373
client	355	isRelatedHandle	377
clientHandle	355	matchForMnemonic	365
clientRectFor	368	maximize	371
close	371	maximizeButton	386
closeId	384	maximized	386
convertToGUIStyle	370	maximizeRect	371
create	379	menuBar	386
deactivateId	384	minimize	371
defaultOrdering	369	minimizeButton	386
defaultPushButton	365	minimized	387
defaultStyle	370	minimizedIcon	387
deferCreation	383	minimizeRect	372
dialogBackground	385	mousePointer	367
dialogBorder	385	moveSizeToClient	368
disabledBackgroundColor	356	nextShellRect	372
dismiss	366	noMoveWithOwner	387

© Copyright IBM Corp. 1992, 1996

IFrameWindow

Member	Page	Member	Page
notifyOwner	366	setRestoreRect	373
registerCallbacks	378	setResult	366
registerFrameClass	382	setToolBarList	374
removeDefaultHandler	382	shareParentDBCSStatus	361
removeExtension	363	shellPosition	387
removeFromWindowList	376	show	374
resetBackgroundColor	356	showModally	367
resetDisabledBackgroundColor	356	sizingBorder	387
restore	372	start	365
restoreRect	372	systemMenu	387
result	366	systemModal	388
setBorderHeight	353	titleBar	388
setBorderSize	354	toolBarList	374
setBorderWidth	355	topHandle	357
setClient	355	tryToLoadDialog	382
setDefaultOrdering	369	unregisterCallbacks	378
setDefaultStyle	370	update	364
setDestroyOnClose	374	useExtensionMinimumSize	364
setExtensions	379	usesDialogBackground	356
setExtensionSize	363	verticalScroll	388
setIcon	352	willDestroyOnClose	374
setLayoutDistorted	375	windowList	388
setMousePointer	367	~IFrameWindow	361

The IFrameWindow class creates and manages frame windows. Typically, these windows are children of the desktop and the parent or owner windows of the User Interface Class Library dialogs and controls. The primary distinction between an IFrameWindow and a generic IWindow (p. 1044) is that frames are conceptually comprised of a number of child windows.

The frame's client window is the control corresponding to the rectangular portion of the frame window not occupied by frame extensions and accessory windows that can be added through frame styles (such as the title bar, system menu, and borders). You do not give the client area an initial size because the client area is automatically sized to fill the frame window.

In addition to the standard frame controls, IFrameWindow supports additional frame extensions. These extensions are primarily application-specific, although IInfoArea (Vol. III) provides a general-purpose information area extension.

You can add extensions to the frame window by providing an IWindow and then specifying the portion of the frame that the IWindow will occupy. The extension can fill a portion of the standard title-bar area or the standard client area.

PM Frame windows include some or all of the following:

- A system menu

- A title bar
- Minimize and maximize buttons
- A border
- A menu bar
- A client

You can construct frame windows from dialog templates loaded from resource libraries. You can then use showModally (p. 367) to display these frame windows as dialogs. In addition, you can use showModally to show frame windows not constructed from dialog templates as application modal.

Win You can construct frame windows from dialog templates loaded from resource libraries. Use WC_DIALOG as the style of the dialog. You can then use showModally (p. 367) to display these frame windows as dialogs.

In addition, you can use showModally to show frame windows not constructed from dialog templates as application modal.

Motif AIX does not support windows constructed from dialog templates. Resource support does not include dialog template support.

Frames include some or all of the following:

- A window menu button
- A title bar
- Minimize and maximize buttons
- A border
- A menu bar
- A client

If you set the titleBar (p. 388) style, the window has a title. If you specify a title, your title is used. If you do not specify a title, the title defaults to the application name.

Frame extensions are only supported in the client area.

The IFrameWindow constructors create an object of this class using several Motif widgets. A TopLevelShell widget provides the frame decorations and window manager communications. An XmMainWindow and an XmForm provide the menu bar support and client area. Frame extensions are supported through attachments on the XmForm widget.

The AIX release of the User Interface Class Library has these characteristics:

- Does not support dialog templates.

© IBM Corp. 1992, 1996

In OS/2 Presentation Manager, the constructors that create IFrameWindow objects from dialog templates create a default frame window if the dialog template is not found. On AIX, these constructors always create default frame windows.

- Treats an IFrameWindow shown using IFrameWindow::showModally (p. 367) as a dialog.
- Does not support clipping a child frame window to a parent frame window.
- Treats a frame window whose parent is not the desktop window or 0 as a secondary window. This frame window does not have minimize or maximize buttons. The parent frame window does not overlay this window (that is, the secondary frame window is raised).

Public Functions

Application Icon

Use these members to manage the application icon used in the frame's system menu and displayed when the frame is minimized.

icon Returns the pointer handle of the window icon.

```
virtual IPointerHandle
  icon() const;
```

Win	PM	Motif
Y	Y	Y

setIcon Use these functions to indicate the icon to use as the application icon.

1
```
virtual IFrameWindow&
  setIcon( const IPointerHandle& icon );
```

Win	PM	Motif
Y	Y	Y

Set the application icon using an existing icon handle.

Motif The Motif Window Manager expects the application icon's size to be 50 pixels by 50 pixels. If you supply an icon with a size other than 50x50 to setIcon, the library scales the icon to 50x50. Because scaling can cause a significant loss of resolution, keep the icon size as close to 50x50 as possible.

2
```
virtual IFrameWindow&
  setIcon( const IResourceId& iconResId );
```

Win	PM	Motif
Y	Y	Y

Set the icon by specifying an IResourceId. If you want to load the icon from a specific resource library, use this function.

Motif The Motif Window Manager expects the application icon's size to be 50 pixels by 50 pixels. If you supply an icon with a size other than 50x50 to setIcon, the library

scales the icon to 50x50. Because scaling can cause a significant loss of resolution, keep the icon size as close to 50x50 as possible.

3 `virtual IFrameWindow&`
`  setIcon( unsigned long iconResId );`

Win	PM	Motif
Y	Y	Y

Set the application icon using an icon from the default resource library.

Motif The Motif Window Manager expects the application icon's size to be 50 pixels by 50 pixels. If you supply an icon with a size other than 50x50 to setIcon, the library scales the icon to 50x50. Because scaling can cause a significant loss of resolution, keep the icon size as close to 50x50 as possible.

Border Size

Use these members to set and query the width and height of the frame window's border.

borderHeight Returns the height of the frame window's top and bottom borders.

```
unsigned long
  borderHeight() const;
```

Win	PM	Motif
Y	Y	I

borderSize Returns the width of the frame window's left and right borders and the height of the frame window's top and bottom borders.

```
ISize
  borderSize() const;
```

Win	PM	Motif
Y	Y	I

Motif In AIX, this function always returns ISize(1,1).

borderWidth Returns the width of the frame window's left and right borders.

```
unsigned long
  borderWidth() const;
```

Win	PM	Motif
Y	Y	I

setBorderHeight

Sets the height of the frame window's top and bottom borders.

```
IFrameWindow&
  setBorderHeight( unsigned long cy );
```

Win	PM	Motif
I	Y	I

cy The height of the top and bottom borders.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidParameter	The system could not set the frame window's border height. The IFrameWindow::sizingBorder (p. 387) style is required for a frame to be able to change its border size.

setBorderSize

Sets the width of the frame window's left and right borders and the height of the frame window's top and bottom borders.

1

```
IFrameWindow&
  setBorderSize( unsigned long cx,
                 unsigned long cy );
```

Win	PM	Motif
I	*Y*	*I*

Use this function to set the width and height of the borders.

cx The width of the left and right borders.
cy The height of the top and bottom borders.

Exceptions	
IInvalidParameter	The system could not set the frame window's border size. The IFrameWindow::sizingBorder (p. 387) style is required for a frame to be able to change its border size.

2

```
IFrameWindow&
  setBorderSize( const ISize& size );
```

Win	PM	Motif
I	*Y*	*I*

Use this function to use an ISize object to set the size of the borders.

size The width of the left and right borders and the height of the top and bottom borders.

Exceptions	
IInvalidParameter	The system could not set the frame window's border size. The IFrameWindow::sizingBorder (p. 387) style is required for a frame to be able to change its border size.

3

```
IFrameWindow&
  setBorderSize( unsigned long cxcy );
```

Win	PM	Motif
I	*Y*	*I*

Use this function to set the borders of the frame to a uniform size.

cxcy The uniform size of all borders.

Exceptions	
IInvalidParameter	The system could not set the frame window's border size. The IFrameWindow::sizingBorder (p. 387) style is required for a frame to be able to change its border size.

setBorderWidth

Sets the width of the frame window's left and right borders.

```
IFrameWindow&
  setBorderWidth( unsigned long cx );
```

Win	PM	Motif
I	*Y*	*I*

cx The width of the left and right borders.

Exceptions	
IInvalidParameter	The system could not set the frame window's border width. The IFrameWindow::sizingBorder (p. 387) style is required for a frame to be able to change its border size.

Client Window

Use these members to manipulate the frame client window. The frame client is similar to the other frame controls, but this control receives notification of certain events not handled by the frame itself. For example, it receives notification for most ICommandEvents (p. 210) that originate from the frame menu bar.

client

Returns a pointer to the IWindow (p. 1044) corresponding to the client window. If no client window exists, or it is not represented by an IWindow object, 0 is returned.

```
virtual IWindow*
  client() const;
```

Win	PM	Motif
Y	*Y*	*Y*

clientHandle

Returns the IWindowHandle (p. 1107) of the client window. If no client window exists, a 0 handle is returned.

```
virtual IWindowHandle
  clientHandle() const;
```

Win	PM	Motif
Y	*Y*	*Y*

setClient

Sets the IWindow (p. 1044) used as the frame window's client area.

```
virtual IFrameWindow&
  setClient( IWindow* newClient );
```

Win	PM	Motif
Y	*Y*	*Y*

newClient The window to set as the frame window's client.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidParameter	The specified window is null. Specify a valid window for the client window.

Color

Use these members to set and query the background color of the frame window.

backgroundColor

Returns the color of the frame window background or the default if no color for the area has been set.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Win Returns the default color of the frame window background.

disabledBackgroundColor

Returns the color of the frame window background when the window is disabled, or the default if no color for the area has been set.

```
virtual IColor
  disabledBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default color of the frame window background when the window is disabled.

resetBackgroundColor

Resets the background color by undoing a previous set.

```
virtual IFrameWindow&
  resetBackgroundColor();
```

Win	PM	Motif
I	*Y*	*N*

resetDisabledBackgroundColor

Resets the disabled background color by undoing a previous set.

```
virtual IFrameWindow&
  resetDisabledBackgroundColor();
```

Win	PM	Motif
I	*Y*	*N*

usesDialogBackground

Returns whether the frame window uses the system dialog background color rather than the system window background color.

```
Boolean
  usesDialogBackground() const;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This function returns false.

Compound Control Functions

These members provide a portable look-and-feel for IFrameWindow. In AIX, the IFrameWindow objects combine various controls to provide the frame window's appearance and behavior. The members provide support for attaching a client, managing frame extensions, and returning the correct window handle for the frame window.

topHandle Returns the top (the oldest) window system control of the IFrameWindow object.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	*N*	*Y*

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IFrameWindow objects because both the copy constructor and assignment operator are private functions.

Three constructors use the given window or resource ID to attempt to create the frame from a dialog template These constructors have the optional parameter *source*. See IFrameWindow::FrameSource (p. 389) for details on this parameter.

The constructors without the *source* do not attempt to load a dialog from a dialog resource. Instead, the frame attributes are specified as additional constructor arguments.

IFrameWindow

1

```
IFrameWindow( const IResourceId& resId,
              IWindow* owner = 0,
              FrameSource source = tryDialogResource );
```

Win	PM	Motif
Y	*Y*	*Y*

Constructs a primary or secondary frame window from a specified resource ID and, optionally, an owner window pointer.

You must specify a resource ID. The owner window pointer defaults to 0. To construct a secondary window, you must specify a nonzero *owner*.

If *source* is IFrameWindow::dialogResource (p. 389), this constructor attempts to load a dialog resource identified by *resId*. If the dialog cannot be loaded, an exception is thrown.

© IBM Corp. 1992, 1996

If *source* is IFrameWindow::noDialogResource (p. 389), this constructor creates a new frame window with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *resId* is used to load the menu bar, accelerator table, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

If *source* is IFrameWindow::tryDialogResource (p. 389), this constructor attempts to load a dialog resource identified by *resId*. If the dialog resource cannot be found, a new frame window is created with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *resId* is used to load the menu bar, accelerator table, title bar text, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

Motif In AIX, this constructor creates a standard frame window.

Do not specify a value for *source*.

2
```
IFrameWindow( unsigned long id = IC_DEFAULT_FRAME_ID,
              FrameSource source = tryDialogResource );
```

Win	PM	Motif
Y	Y	Y

Constructs a primary frame window with the specified window identifier.

If *source* is IFrameWindow::dialogResource (p. 389), this constructor attempts to load a dialog resource identified by *id*. If the dialog cannot be loaded, an exception is thrown.

If *source* is IFrameWindow::noDialogResource (p. 389), this constructor creates a new frame window with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *id* is used to load the menu bar, accelerator table, title bar text, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

If *source* is IFrameWindow::tryDialogResource (p. 389), this constructor attempts to load a dialog resource identified by *id*. If the dialog resource cannot be found, a new frame window is created with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *id* is used to load the menu bar, accelerator table, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

Motif In AIX, this constructor creates a standard frame window.

Do not specify a value for *source*.

3
```
IFrameWindow( const IResourceId& resId,
              IWindow* parent,
              IWindow* owner,
              FrameSource source = tryDialogResource );
```

Win	PM	Motif
Y	Y	Y

Constructs a child window from a specified resource ID, parent window, and owner window.

If *source* is IFrameWindow::dialogResource (p. 389), this constructor attempts to load a dialog resource identified by *resId*. If the dialog cannot be loaded, an exception is thrown.

If *source* is IFrameWindow::noDialogResource (p. 389), this constructor creates a new frame window with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *resId* is used to load the menu bar, accelerator table, title bar text, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

If *source* is IFrameWindow::tryDialogResource (p. 389), this constructor attempts to load a dialog resource identified by *resId*. If the dialog resource cannot be found, a new frame window is created with the style returned by IFrameWindow::defaultStyle (p. 370). The resource library identified by *resId* is used to load the menu bar, accelerator table, and icon for the frame if IFrameWindow::defaultStyle returns these styles.

Motif In AIX, this constructor creates a standard frame window.

Do not specify a value for *source*.

4 `IFrameWindow( const IWindowHandle& hwnd );`

Win	PM	Motif
Y	Y	Y

Construct an IFrameWindow object from the window handle of an existing frame window.

PM This constructor permits you to migrate from existing OS/2 Presentation Manager code that creates the frame window directly. This constructor wraps an existing Presentation Manager frame window (created outside the User Interface Class Library) with an IFrameWindow C++ object.

Motif AIX does not support this constructor.

5
```
IFrameWindow( const IResourceId& resId,
              const IFrameWindow::Style& style );
```

Win	PM	Motif
Y	Y	Y

Construct an IFrameWindow object from a specified resource ID and library, and frame window style.

Use this constructor if your frame window is not a dialog and requires frame resources, such as a menu bar.

© IBM Corp. 1992, 1996

6 `IFrameWindow( const IFrameWindow::Style& style, const IResourceId& resId = IC_DEFAULT_FRAME_ID );`

Win	PM	Motif
Y	Y	Y

Construct an IFrameWindow object from a frame style and, optionally, a resource ID.

Use this constructor to customize the style of a frame window without having to specify a resource ID. If you do not specify *resID*, the constructor uses the default resource ID IC_DEFAULT_FRAME_ID that is defined in <ICCONST.H>.

7
```
IFrameWindow(
   const char* title,
   const IResourceId& resId = IC_DEFAULT_FRAME_ID,
   const IFrameWindow::Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Construct an IFrameWindow from a title, and optionally, a resource ID and frame style.

Use this constructor to customize the title for a frame window without specifying a resource ID or frame style. If you do not specify *resId*, the constructor uses the default resource ID IC_DEFAULT_FRAME_ID that is defined in <ICCONST.H>. If you do not specify *style*, the constructor calls IFrameWindow::defaultStyle (p. 370) to return the default style.

8
```
IFrameWindow(
   const IResourceId& resId,
   IWindow* parent,
   IWindow* owner,
   const IRectangle& initRect,
   const IFrameWindow::Style& style = defaultStyle ( ),
   const char* title = 0 );
```

Win	PM	Motif
Y	Y	Y

Construct an IFrameWindow object from a full set of frame attributes: a resource ID, parent, owner, initial rectangle (position and size), and, optionally, a style and title.

Use this constructor when you want to supply the values instead of using the default values, such as initializing the size and title. This constructor gives you full control over the attributes of nondialog frame windows. If you do not specify *style*, the constructor calls IFrameWindow::defaultStyle (p. 370) to return the default style.

The default for *title* is 0. If you set the style IFrameWindow::titleBar (p. 388), the frame has a title. If you supply a nonzero parameter, the title is the specified string. Otherwise, the constructor attempts to load a string from the resource library with the frame window's ID. If the attempt fails, the constructor uses a default title, typically the name of the executable file.

~IFrameWindow

```
virtual
  ~IFrameWindow();
```

Win	PM	Motif
Y	Y	Y

DBCS Support

DBCS support members share DBCS information between a parent and child frame.

shareParentDBCSStatus

Causes a child frame to share the DBCS status control of its parent. If either the child or parent do not have the style IFrameWindow::appDBCSStatus (p. 385) set, this function has no effect.

```
virtual IFrameWindow&
  shareParentDBCSStatus();
```

Win	PM	Motif
N	Y	I

Frame Extensions

Use these members to add extensions to the standard frame window. Such extensions are simply additional windows placed in specific locations. The information area implemented by the IInfoArea (Vol. III) class is an example of such an extension.

addExtension

Adds a window as a frame extension.

1
```
virtual IFrameWindow&
  addExtension( IWindow* newExtension,
                Location location,
                double percentage,
                SeparatorType separator = thinLine );
```

Win	PM	Motif
Y	Y	Y

Use this function to add a relative type of frame extension.

newExtension
The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, or client area) to be occupied by the frame extension. Use the enumeration Location (p. 389) to specify the location for the frame extension.

percentage The portion of the frame window area allocated to the frame extension, in a floating point percentage ($0 < x < 1.0$).

© IBM Corp. 1992, 1996

separator The separator type. Use the enumeration SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

2

```
virtual IFrameWindow&
  addExtension( IWindow* newExtension,
                Location location,
                unsigned long widthOrHeight,
                SeparatorType separator = thinLine );
```

Win	PM	Motif
Y	*Y*	*Y*

Use this function when you want the frame extension to be a fixed width or height.

newExtension
The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, or client area) to be occupied by the frame extension. Use the enumeration Location (p. 389) to specify the location for the frame extension.

widthOrHeight
The portion of the frame window area allocated to the frame extension, in a fixed width or height.

separator The separator type. Use the enumeration SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

3

```
virtual IFrameWindow&
  addExtension( IWindow* newExtension,
                Location location,
                int widthOrHeight,
                SeparatorType separator = thinLine );
```

Win	PM	Motif
Y	*Y*	*N*

Use this function when you want the frame extension to be a fixed width or height.

newExtension
The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, or client area) to be occupied by the frame extension. Use the enumeration Location (p. 389) to specify the location for the frame extension.

widthOrHeight
The portion of the frame window area allocated to the frame extension, in a fixed width or height.

separator The separator type. Use the enumeration SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

4
```
virtual IFrameWindow&
  addExtension( IWindow* newExtension,
                Location location,
                SeparatorType separator = thinLine );
```

Win	PM	Motif
Y	Y	N

Use this function when you want the frame extension to use the minimum size of the associated control.

newExtension
The frame extension window.

location The relative location (left, right, top, or bottom) and frame window area (title bar area, menu bar area, or client area) to be occupied by the frame extension. Use the enumeration Location (p. 389) to specify the location for the frame extension.

separator The separator type. Use the enumeration SeparatorType (p. 390) to specify the type of separator delimiting the frame extension.

isAnExtension

Returns true if the specified IWindow object is a frame extension.

```
virtual Boolean
  isAnExtension( const IWindow* window ) const;
```

Win	PM	Motif
Y	Y	Y

removeExtension

Removes a frame extension (window).

```
virtual IFrameWindow&
  removeExtension( IWindow* extension,
                   Boolean updateDisplay = true );
```

Win	PM	Motif
Y	Y	Y

setExtensionSize

Sets a frame extension to the specified size.

1
```
virtual IFrameWindow&
  setExtensionSize( IWindow* extension,
                    double widthOrHeight );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the size of a fixed-type extension.

© IBM Corp. 1992, 1996

extension The frame extension window.

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a floating point percentage (double, 0 < x < 1.0).

2
```
virtual IFrameWindow&
  setExtensionSize( IWindow* extension,
                    unsigned long widthOrHeight );
```

Win	PM	Motif
Y	*Y*	*Y*

Use this function to set the size of a fixed-type extension.

extension The frame extension window.

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a fixed width or height (unsigned long).

3
```
virtual IFrameWindow&
  setExtensionSize( IWindow* extension,
                    int widthOrHeight );
```

Win	PM	Motif
Y	*Y*	*N*

Use this function to set the size of a fixed-type extension.

extension The frame extension window.

widthOrHeight
The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in a fixed width or height (int).

update

Reconfigures all of the controls and extensions for the frame window. The functions IFrameWindow::addExtension (p. 361) and IFrameWindow::setClient (p. 355) automatically call this function.

```
virtual IFrameWindow&
  update();
```

Win	PM	Motif
Y	*Y*	*Y*

useExtensionMinimumSize

Sizes the width or height of the frame extension based on the minimum size of the window it contains.

```
virtual IFrameWindow&
  useExtensionMinimumSize( IWindow* extension );
```

Win	PM	Motif
Y	Y	N

Implementation

These members create the frame window. The loading of a dialog window, creating a frame window, and adding default handlers are handled by members in this group. A derived class can override these functions to customize frame-window processing. For example, a derived class can use the IFrameWindow::deferCreation (p. 383) style to indicate that it will create the frame window.

start

Creates a frame handler and adds it to the (newly created) frame window.

```
IFrameWindow&
  start( const IWindowHandle& hwnd );
```

Win	PM	Motif
Y	Y	Y

hwnd The frame window handle.

Layout Support

Layout support members supply information used by the IFrameWindow class to provide dialog-like behavior. These members override members of IWindow.

defaultPushButton

Returns the current default push button if this frame window is in its parent window chain.

Pressing the Enter key causes the default push button to be selected.

```
virtual IWindowHandle
  defaultPushButton() const;
```

Win	PM	Motif
Y	N	N

matchForMnemonic

Returns the first child window using the specified character as a mnemonic.

```
virtual IWindowHandle
  matchForMnemonic( unsigned short character ) const;
```

Win	PM	Motif
Y	N	N

Modal Display

Use these members to manage frame windows (dialogs) in an application-modal fashion. When you display an *application-modal* frame window, its owner window becomes disabled. The user

© IBM Corp. 1992, 1996

cannot interact with the owner window until first dismissing the application-modal frame window.

dismiss Closes a frame window by hiding it and, if modal, completes the calls to IFrameWindow::showModally (p. 367), which triggered the window's display.

```
virtual IFrameWindow&
  dismiss( unsigned long result = 0 );
```

Win	PM	Motif
Y	Y	Y

result Value for this use of the frame window. The default is 0.

notifyOwner Posts a command event to the frame window's (or dialog's) owner.

```
virtual IFrameWindow&
  notifyOwner(
   unsigned long id,
   ICommandEvent::Source source = ICommandEvent::unknown,
   Boolean pointerDevice = false );
```

Win	PM	Motif
Y	Y	Y

id The command ID. The meaning of the command ID depends on the *source* of the command. The command ID can be a push button ID, a menu item ID, an accelerator command value, or other control-specific command information.

source The origin of the command event. Use the enumeration ICommandEvent::Source (p. 213) to specify the origin of the command event. The default is the enumerator ICommandEvent::unknown, which indicates control-specific command information.

pointerDevice
Pointer device indicator. If true, the command is the result of a pointer-device operation. If false, the command is the result of a keyboard operation. The default is false.

result Returns the frame result value. You set this value previously by calling either IFrameWindow::setResult (p. 366) or IFrameWindow::dismiss (p. 366).

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

Win	PM	Motif
Y	Y	Y

setResult Sets the result value for the frame. This is typically used to indicate what action the user selected on a modal dialog. For example, if the user selects an **OK** button, you call setResult(DID_OK). If the user selects a **Cancel** button, you call setResult(DID_CANCEL). Subsequently, you can query the result to see what action the user requested using IFrameWindow::result (p. 366). For example:

```
IFrameWindow dialog( MY_DIALOG );
dialog.showModally();
if ( dialog.result() == DID_OK )
  // User said to go ahead...
else
  // User changed his mind...
```

```
virtual IFrameWindow&
  setResult( unsigned long result );
```

Win	PM	Motif
Y	Y	Y

result The result value for this use of the frame window.

showModally Displays the frame window modally. While the frame window is being shown modally, its parent or owner or both windows are disabled. Other top level windows belonging to the application are not disabled.

```
virtual unsigned long
  showModally();
```

Win	PM	Motif
Y	Y	Y

PM Modal frame windows do not receive system command events with a command ID of ISystemMenu::idClose (SC_CLOSE) when they are closed from the task list. Frame windows that require this close notification should not be shown modally or should not be added to the task list.

Exceptions	
IInvalidParameter	The frame window's owner and parent windows must not be the same when the frame is shown modally.

Mouse Pointer

Use these members to manage the mouse pointer used in the frame window.

mousePointer

Returns the handle of the mouse pointer used when the mouse is over the frame window. If the handle is 0, the default mouse pointer is used.

```
virtual IPointerHandle
  mousePointer() const;
```

Win	PM	Motif
Y	Y	N

setMousePointer

Sets the mouse pointer to be used when the mouse is over the frame window. If you specify a null mouse pointer handle, the default mouse pointer is used.

© IBM Corp. 1992, 1996

The mouse pointer change will not take effect until related events on the queue are processed. This happens, for example, when all handlers currently processing an event have returned, or when a dialog is displayed modally.

```
virtual IFrameWindow&
  setMousePointer( const IPointerHandle& mousePointer );
```

Win	PM	Motif
Y	Y	N

Moving and Sizing

Use these members to move and size the frame and its client window.

clientRectFor

Returns a rectangle indicating the size and position the client window has if the frame is sized and positioned according to the specified rectangle.

```
virtual IRectangle
  clientRectFor( const IRectangle& frameRect ) const;
```

Win	PM	Motif
Y	Y	Y

frameRect The proposed size and position of the frame.

frameRectFor

Returns a rectangle indicating the frame window's size and position required to produce the specified client window.

```
virtual IRectangle
  frameRectFor( const IRectangle& clientRect ) const;
```

Win	PM	Motif
Y	Y	Y

clientRect The proposed client window's size and position.

Exceptions	
IAccessError	The system could not calculate the frame window's rectangle. Possible causes include an invalid window handle was specified or the calculated rectangle was empty.

moveSizeToClient

Sizes and positions the frame window around the specified client window's rectangle.

```
virtual IFrameWindow&
  moveSizeToClient( const IRectangle& clientRect );
```

Win	PM	Motif
Y	Y	Y

clientRect The client window's rectangle for the frame window to contain.

Exceptions	
IAccessError	The system could not calculate the frame window's rectangle. Possible causes include an invalid window handle was specified or the calculated rectangle was empty.

Observer Notification

Observer notification members implement the INotifier (Vol. I) protocol for IWindow (p. 1044) classes.

enableNotification

Enables the frame window to send notifications to any observer objects.

```
virtual IFrameWindow&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

enable Enablement flag. If true, notification is enabled; if false, notification is disabled.

Sibling Order

These members affect the ordering of windows with their siblings. The window order (or Z-order) is used when tabbing or painting sibling windows. Window painting starts from the bottom sibling and proceeds to the topmost sibling window. Cursor tabbing occurs from the topmost sibling and proceeds to the bottom sibling window.

defaultOrdering

Returns the order in which new windows are created relative to their sibling windows.

```
static IWindow::SiblingOrder
  defaultOrdering();
```

Win	PM	Motif
Y	Y	Y

setDefaultOrdering

Specifies whether the User Interface Class Library creates new windows on top of their sibling windows or behind them. By default, the library creates windows behind their other siblings.

```
static void
  setDefaultOrdering( IWindow::SiblingOrder ordering );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

ordering Use the enumeration IWindow::SiblingOrder (p. 1099) to specify the order of sibling window creation. If you specify onTopOfSiblings, the library creates new windows on top of their siblings.

Standard Control Access

Use these members to get information about the standard frame controls.

handleFor

Returns the IWindowHandle (p. 1107) corresponding to a standard frame control. If the control is not present, 0 is returned.

```
virtual IWindowHandle
  handleFor( const Style& standardControl ) const;
```

Win	PM	Motif
Y	Y	Y

Motif In AIX, this function always returns 0.

Styles

IFrameWindow defines objects of the nested class IFrameWindow::Style (p. 392). You can use these styles with IFrameWindow::setDefaultStyle (p. 370) and with constructors for the IFrameWindow class. You can combine IFrameWindow::Style objects with the objects of the class IWindow::Style (p. 1093).

convertToGUIStyle

Converts style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles that are defined by the User Interface Class Library can be returned by setting the *extendedOnly* parameter to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. This style is IFrameWindow::classDefaultStyle (p. 385) unless you have changed the style using IFrameWindow::setDefaultStyle (p. 370).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent frame windows.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

style Use objects of the classes IFrameWindow::Styles (p. 383) and IWindow::Styles (p. 1093) to specify the default style.

System-Menu-Related Actions

Use these members with frame operations related to the actions on the system menu.

These operations include minimizing and maximizing a frame window. You can also restore a frame window after it has been minimized or maximized. You can query the rectangle value for a maximized or minimized window or the rectangle to which the frame window is restored after minimizing or maximizing the frame.

You can close the frame window or query the recommended rectangle value for the next frame window.

close

Closes the frame window. Closing a frame window causes all secondary and child windows to close. Closing the last primary window in your application causes the event (message) queue to terminate. When the queue terminates, the run member function in your main routine ends.

```
virtual IFrameWindow&
  close();
```

Win	PM	Motif
Y	Y	Y

maximize

If the window has a maximize button, this function maximizes the window. If the window does not have a maximize button, this function does nothing.

```
virtual IFrameWindow&
  maximize();
```

Win	PM	Motif
Y	Y	I

maximizeRect

Returns a rectangle indicating the size and position the frame window has when it is maximized.

```
virtual IRectangle
  maximizeRect() const;
```

Win	PM	Motif
Y	Y	I

Motif AIX does not support this function; therefore, it returns a default IRectangle.

Exceptions	
IAccessError	The system could not determine the frame window's maximized position. A possible cause is an invalid window handle.

minimize

Minimizes the window. If the window does not have a minimize button, this function does nothing.

© IBM Corp. 1992, 1996

```
virtual IFrameWindow&
  minimize();
```

Win	PM	Motif
Y	Y	Y

minimizeRect Returns a rectangle indicating the size and position the frame window has when it is minimized.

```
virtual IRectangle
  minimizeRect() const;
```

Win	PM	Motif
Y	Y	I

Win The position of the rectangle returned is not valid until the window is or has been actually minimized.

If you are using the Windows 95 desktop, you should ignore the position of the rectangle returned at all times. The size of the rectangle is the default size of items placed into the taskbar.

Motif AIX does not support this function; therefore, it returns a default IRectangle.

Exceptions	
IAccessError	The system could not determine the frame window's minimized position. A possible cause is an invalid window handle.

nextShellRect

Returns the system-recommended coordinates of the next main window.

```
static IRectangle
  nextShellRect();
```

Win	PM	Motif
Y	Y	I

Motif In AIX, this function always returns an IRectangle with a default location of IPoint(0,0) and a default size of 1/3 screen height by 1/3 screen width.

Exceptions	
IAccessError	The system could not generate recommended values for the size and position of the frame window. A possible cause is an invalid anchor block for the current thread.

restore Restores a maximized or minimized window to its previous size and position.

```
virtual IFrameWindow&
  restore();
```

Win	PM	Motif
Y	Y	I

Motif This function only restores a window from its minimized state.

restoreRect Returns the window rectangle coordinates to which the window will be restored.

```
virtual IRectangle
  restoreRect() const;
```

Win	PM	Motif
Y	Y	I

Motif This function returns a default IRectangle object.

setRestoreRect

Sets the rectangle coordinates indicating the size and position to which the window is restored. If the size of the specified rectangle is larger than the size of the screen, the actual restore rectangle set is the size of the screen.

```
virtual IFrameWindow&
  setRestoreRect( const IRectangle& rect );
```

Win	PM	Motif
Y	Y	I

rect The rectangle coordinates to restore the window to.

Testing

Use these members to test various attributes of the frame window. These attributes include whether the frame is flashing, the frame is minimized or maximized, or the frame window has been shown modally.

isFlashing

If the frame window is flashing, true is returned.

```
Boolean
  isFlashing() const;
```

Win	PM	Motif
Y	Y	I

Motif In AIX, this function always returns false.

isMaximized

If the frame window is maximized, true is returned.

```
Boolean
  isMaximized() const;
```

Win	PM	Motif
Y	Y	I

Motif In AIX, this function always returns false.

isMinimized

If the frame window is minimized, true is returned.

```
Boolean
  isMinimized() const;
```

Win	PM	Motif
Y	Y	Y

isModal

If the window is displayed in application-modal mode, true is returned.

```
Boolean
  isModal() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Tool Bar Support

Use these members to manage the use of tool bar containers in a frame window.

setToolBarList

Stores the address of a list containing the tool bars used in this frame window.

```
IFrameWindow&
  setToolBarList( IToolBarList* toolBarList );
```

Win	PM	Motif
Y	Y	N

toolBarList

Returns the address of the list of tool bars used in this frame window.

```
IToolBarList*
  toolBarList() const;
```

Win	PM	Motif
Y	Y	N

Window Destruction

These members destroy the GUI window associated with an IFrameWindow object when the window is closed.

setDestroyOnClose

Sets the destroy window flag on or off. If the flag is on, the window object is destroyed when the window is closed.

```
virtual IFrameWindow&
  setDestroyOnClose( Boolean destroy = true );
```

Win	PM	Motif
Y	Y	Y

destroy The destroy window flag's value. The default value is true (on).

willDestroyOnClose

Returns the state of the destroy window flag. The default state is on.

```
Boolean
  willDestroyOnClose() const;
```

Win	PM	Motif
Y	Y	Y

Window Display

Use the window display members to manage the visibility of the frame window.

show

Shows or hides the frame window. If the flag is true, the frame window is shown. If the flag is false, the frame window is hidden.

```
virtual IFrameWindow&
  show( Boolean showWindow = true );
```

Win	PM	Motif
Y	Y	Y

Window Flashing

Use these members to make the frame start and stop flashing. You could flash the frame window to alert the user to an error situation.

beginFlashing

Starts flashing the frame. Use this to alert the user to something that requires immediate attention.

```
virtual IFrameWindow&
  beginFlashing();
```

Win	PM	Motif
Y	Y	I

Exceptions	
IAccessError	The system could not flash the frame window. A possible cause is an invalid window handle.

endFlashing

Stops flashing the frame window.

```
virtual IFrameWindow&
  endFlashing();
```

Win	PM	Motif
Y	Y	I

Exceptions	
IAccessError	An error occurred when the system tried to stop flashing the frame window. A possible cause is an invalid frame window handle.

Window Layout

Window layout members notify the frame of changes to its layout.

setLayoutDistorted

Indicates that changes have occurred in the window that will cause the layout of the window to be updated.

```
IFrameWindow&
  setLayoutDistorted( unsigned long layoutAttributesOn,
                      unsigned long layoutAttributesOff );
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

Window List

Use these members to add and remove frame titles from the window list.

addToWindowList

Adds this frame window's title as a window list entry. If you construct this frame window with the style IFrameWindow::windowList (p. 388), the title is automatically added to the window list.

```
virtual IFrameWindow&
  addToWindowList();
```

Win	PM	Motif
Y	*Y*	*I*

Exceptions	
IAccessError	The frame window's title was not added as an entry in the window list. Possible causes include the following: a title bar does not exist for the frame window, the limit has been reached for number of window list entries, or the frame window had a previous window list entry that was invalid.

removeFromWindowList

Removes this frame's entry from the window list.

```
virtual IFrameWindow&
  removeFromWindowList();
```

Win	PM	Motif
Y	*Y*	*I*

Exceptions	
IAccessError	The system could not remove the entry from the window list. A possible cause is an invalid window list handle.

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Compound Control Functions

These members provide a portable look-and-feel for IFrameWindow. In AIX, the IFrameWindow objects combine various controls to provide the frame window's appearance and behavior. The members provide support for attaching a client, managing frame extensions, and returning the correct window handle for the frame window.

attachClient Attaches the client control to a frame extension of the frame window.

p The frame extension that the client control is to be attached to.

```
IFrameWindow&
  attachClient( IFrameExtension* p );
```

Win	PM	Motif
N	*N*	*Y*

findExtensions

Searches the frame extensions collection for the last extensions added for the following locations: above, below, left, and right of the client.

```
IFrameWindow&
  findExtensions( IFrameExtension *& topExt,
                  IFrameExtension *& bottomExt,
                  IFrameExtension *& leftExt,
                  IFrameExtension *& rightExt );
```

Win	PM	Motif
N	*N*	*Y*

isFrameWindow

If this is an IFrameWindow object, true is returned.

```
virtual Boolean
  isFrameWindow() const;
```

Win	PM	Motif
N	*N*	*Y*

isRelatedHandle

If the specified handle is one of the controls IFrameWindow created during construction, true is returned.

```
virtual Boolean
  isRelatedHandle( const IWindowHandle& windowHandle ) const;
```

Win	PM	Motif
N	*N*	*Y*

© IBM Corp. 1992, 1996

Event-Handling Implementation

Event-handling implementation members perform processing that allows handlers to receive GUI events and to route these events.

registerCallbacks

Registers Motif callbacks for the widgets created during object construction of this class.

Registers all possible callbacks and X event handlers to this object for events it might receive. IHandler derived classes later determine which events they will process.

If classes you derive override the registerCallbacks member function, the override must call Inherited::registerCallbacks to register the applicable callbacks and X event handlers.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

Exceptions	
IInvalidParameter	The application callbacks could not be registered. A possible cause is an uninitialized environment.

unregisterCallbacks

Unregisters the Motif callbacks for the underlying widgets. This function is called during the destruction of the IFrameWindow.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Frame Extensions Management

These protected members maintain information about the frame window's collection of frame extensions. You can query the location of a specific frame extension within the frame extension collection.

extensions

Returns a pointer to the IFrameExtensions (p. 338) collection for the frame window.

```
IFrameExtensions*
  extensions() const;
```

Win	PM	Motif
Y	Y	Y

findExtension

Returns the index of the frame extension corresponding to the specified control IWindow (p. 1044). The index locates the IFrameExtension (p. 329) in the IFrameExtensions (p. 338) collection of the frame window.

```
unsigned
  findExtension( IWindow* window );
```

Win	PM	Motif
Y	Y	Y

window The window associated with the frame extension.

Exceptions	
IInvalidParameter	The specified control did not match any currently defined frame extension's control. The specified control has not been defined as an frame extension.

setExtensions

Sets a pointer to the IFrameExtensions (p. 338) collection for the frame window.

```
IFrameWindow&
  setExtensions( IFrameExtensions* extensions );
```

Win	PM	Motif
Y	Y	Y

extensions The pointer to the collection of frame extensions.

Implementation

These members create the frame window. The loading of a dialog window, creating a frame window, and adding default handlers are handled by members in this group. A derived class can override these functions to customize frame-window processing. For example, a derived class can use the IFrameWindow::deferCreation (p. 383) style to indicate that it will create the frame window.

addDefaultHandler

Adds the default frame handler to a newly created frame.

```
IFrameWindow&
  addDefaultHandler();
```

Win	PM	Motif
Y	Y	Y

create

Creates a frame window.

© IBM Corp. 1992, 1996

IFrameWindow

Win	PM	Motif
N	N	Y

```
IWindowHandle
  create(
   unsigned long id,
   const char* title,
   unsigned long style,
   IXmCreateFunction createFunction,
   const IWindowHandle& parent,
   const IWindowHandle& owner,
   const IRectangle& initRect,
   const void* callerArgList,
   const unsigned int callerNumberArguments,
   IWindow::SiblingOrder ordering = defaultOrdering ( ) );
```

This function is AIX-specific. You can specify a create widget function for the widget type.

id The frame window's resource ID.

title The frame window's title.

style The default style specification for the frame window. Use the styles provided by IFrameWindow::Styles (p. 383) and IWindow::Styles (p. 1093) to specify the default style.

createFunction
The create widget function for the widget type

parent The parent of the frame window.

owner The owner of the frame window.

initRect The initial frame window rectangle, defined by IFrameWindow::nextShellRect (p. 372).

callerArgList
The resource argument list for the widget type.

callerNumberArguments
The number of resource arguments in the *callerArgList*.

ordering Sibling window order. Use the enumeration IWindow::SiblingOrder (p. 1099) to specify the order of sibling window creation.

Exceptions	
IInvalidParameter	A valid creation function must be specified on the call to create.

2

```
IWindowHandle
  create(
   unsigned long id,
   const char* title,
   unsigned long style,
   const char* windowClass,
   const IWindowHandle& parent,
   const IWindowHandle& owner,
   const IRectangle& initRect,
   const void* ctlData,
   const void* presParams,
   IWindow::SiblingOrder ordering = defaultOrdering ( ),
   unsigned long extendedStyle = 0 );
```

Win	PM	Motif
Y	*Y*	*N*

This function creates the frame window.

id The frame window's resource ID.

title The frame window's title.

style The default style specification for the frame window. Use the styles provided by IFrameWindow::Styles (p. 383) and IWindow::Styles (p. 1093) to specify the default style.

windowClass
The window class of the frame window.

parent The parent of the frame window.

owner The owner of the frame window.

initRect The initial frame window rectangle, defined by IFrameWindow::nextShellRect (p. 372).

ctlData Control data for the window.

presParams
Presentation parameters for the window.

ordering Sibling window order. Use the enumeration IWindow::SiblingOrder (p. 1099) to specify the order of sibling window creation.

extendedStyle
The extended style for the frame window (Windows only).

initialize Initializes the object by creating a standard frame window.

© IBM Corp. 1992, 1996

IFrameWindow

```
IFrameWindow&
  initialize( const IResourceId& resId,
              const Style& style,
              IWindow* parent = 0,
              IWindow* owner = 0,
              const IRectangle& initRect = nextShellRect ( ),
              const char* title = 0 );
```

Win	PM	Motif
Y	*Y*	*I*

resId The resource identifier of the frame window.

style The default style specification for the frame window. Use the styles provided by IFrameWindow::Styles (p. 383) and IWindow::Styles (p. 1093) to specify the default style.

parent The parent of the frame window.

owner The owner of the frame window.

initRect The initial frame window rectangle, defined by IFrameWindow::nextShellRect (p. 372).

title The title of the frame window.

registerFrameClass

Registers the window class for the frame.

```
unsigned long
  registerFrameClass( const Style& style,
                      const IResourceId& resId );
```

Win	PM	Motif
Y	*N*	*N*

removeDefaultHandler

Removes the default frame handler from the frame window. If the default handler is attached to the frame window, the destructor automatically calls this function.

```
IFrameWindow&
  removeDefaultHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

tryToLoadDialog

Attempts to load a dialog window from a dialog template resource if *source* is IFrameWindow::dialogResource. If *source* is IFrameWindow::noDialogResource, or *source* is IFrameWindow::tryDialogResource and the dialog cannot be loaded, this function calls IFrameWindow::initialize (p. 381).

```
IFrameWindow&
  tryToLoadDialog( const IResourceId& resId,
                   IWindow* parent = 0,
                   IWindow* owner = 0,
                   FrameSource source = tryDialogResource );
```

Win	PM	Motif
Y	Y	Y

resId The resource identifier of the frame window.

parent The parent of the frame window.

owner The owner of the frame window.

source Specifies if the frame window should be constructed by loading a dialog template or created using the styles returned by IFrameWindow::defaultStyle (p. 370).

Motif AIX does not support this function; therefore, it has no effect.

Exceptions	
IAccessError	The frame window could not be created from the specified dialog template. Verify that the specified owner and parent window are valid.
IInvalidParameter	*source* did not contain one of the FrameSource (p. 389) enumerator values.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Implementation

These members create the frame window. The loading of a dialog window, creating a frame window, and adding default handlers are handled by members in this group. A derived class can override these functions to customize frame-window processing. For example, a derived class can use the IFrameWindow::deferCreation (p. 383) style to indicate that it will create the frame window.

deferCreation

When a derived class creates the frame window, the derived class specifies this style. For example, a derived class could call IFrameWindow::initialize (p. 381), IFrameWindow::create (p. 379), or IFrameWindow::tryToLoadDialog (p. 382) to create a frame window.

© IBM Corp. 1992, 1996

```
static const Style
  deferCreation;
```

Win	PM	Motif
Y	Y	Y

Notification Event Descriptions

Use these INotificationId strings for all notifications that IFrameWindow provides to its observers.

activateId Notification identifier provided to observers when the frame window is activated.

```
static INotificationId const
  activateId;
```

Win	PM	Motif
Y	Y	Y

closeId Notification identifier provided to observers when the frame window is closed.

```
static INotificationId const
  closeId;
```

Win	PM	Motif
Y	Y	Y

deactivateId Notification identifier provided to observers when the frame window is deactivated.

```
static INotificationId const
  deactivateId;
```

Win	PM	Motif
Y	Y	Y

Styles

IFrameWindow defines objects of the nested class IFrameWindow::Style (p. 392). You can use these styles with IFrameWindow::setDefaultStyle (p. 370) and with constructors for the IFrameWindow class. You can combine IFrameWindow::Style objects with the objects of the class IWindow::Style (p. 1093).

accelerator Associates an accelerator key table with the frame window. The frame window constructor loads the table from its resource library.

```
static const Style
  accelerator;
```

Win	PM	Motif
Y	Y	Y

alignNoAdjust

Restricts adjustment of the frame position. If you specify this style, you lose performance optimizations provided by frame position adjustment.

```
static const Style
  alignNoAdjust;
```

Win	PM	Motif
Y	Y	Y

animated Shows the frame window with animation when it is opened, closed, or restored.

```
static const Style
  animated;
```

Win	PM	Motif
I	*Y*	*Y*

appDBCSStatus

Includes a DBCS status area in the frame window when it is displayed in a DBCS environment.

Note: If you use a DBCS-operating system or machine, there is always one status area, even if you do not specify it.

```
static const Style
  appDBCSStatus;
```

Win	PM	Motif
I	*Y*	*Y*

border Puts a border around the frame window.

```
static const Style
  border;
```

Win	PM	Motif
Y	*Y*	*Y*

classDefaultStyle

Specifies the original default style for this class, which is IFrameWindow::titleBar (p. 388) | IFrameWindow::systemMenu (p. 387) | IFrameWindow::minimizeButton (p. 386) | IFrameWindow::windowList (p. 388) | IFrameWindow::maximizeButton (p. 386) | IFrameWindow::sizingBorder (p. 387) | IFrameWindow::appDBCSStatus (p. 385).

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

dialogBackground

Gives the system dialog's background color to a frame that was not created from a dialog template. Without this style, the frame has the system window background color.

```
static const Style
  dialogBackground;
```

Win	PM	Motif
Y	*Y*	*Y*

dialogBorder Puts a dialog border around the frame window.

```
static const Style
  dialogBorder;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

hideButton Gives the frame window a hide button.

```
static const Style
  hideButton;
```

Win	PM	Motif
I	*Y*	*Y*

PM This style is mutually exclusive with the IFrameWindow::minimizeButton (p. 386) style.

Win This style is interpreted exactly like the minimizeButton (p. 386) style. If you are using the Windows 95 desktop, setting this style will cause the systemMenu (p. 387) style to be set automatically.

horizontalScroll

Gives the frame window a horizontal scroll bar.

```
static const Style
  horizontalScroll;
```

Win	PM	Motif
Y	*Y*	*Y*

maximizeButton

Gives the frame window a maximize button.

```
static const Style
  maximizeButton;
```

Win	PM	Motif
Y	*Y*	*Y*

Win If you are using the Windows 95 desktop, setting this style will cause the systemMenu (p. 387) style to be set automatically.

maximized Creates the frame window in the maximized state.

```
static const Style
  maximized;
```

Win	PM	Motif
Y	*Y*	*Y*

menuBar Gives the frame window a menu bar, which is loaded from the application's resource library. If you want to construct the frame window dynamically instead of from a resource library, use the class IMenuBar (p. 554) instead of this style.

```
static const Style
  menuBar;
```

Win	PM	Motif
Y	*Y*	*Y*

minimizeButton

Gives the frame window a minimize button. This style is mutually exclusive with the IFrameWindow::hideButton (p. 386) style.

```
static const Style
  minimizeButton;
```

Win	PM	Motif
Y	Y	Y

Win If you are using the Windows 95 desktop, setting this style will cause the systemMenu (p. 387) style to be set automatically.

minimized Causes the frame window to be created in the minimized state.

```
static const Style
  minimized;
```

Win	PM	Motif
Y	Y	Y

minimizedIcon

Associates an icon with the frame window. The system uses this icon when it minimizes the frame window.

```
static const Style
  minimizedIcon;
```

Win	PM	Motif
Y	Y	Y

noMoveWithOwner

Disables the frame window's default behavior of automatically moving with its owner.

```
static const Style
  noMoveWithOwner;
```

Win	PM	Motif
I	Y	Y

Win This style is always set, meaning that frame windows do not move with their owner.

shellPosition Sets the initial position of the window to a location and size dictated by the system shell.

```
static const Style
  shellPosition;
```

Win	PM	Motif
Y	Y	Y

sizingBorder Puts a sizing border around the frame window.

```
static const Style
  sizingBorder;
```

Win	PM	Motif
Y	Y	Y

systemMenu Gives the frame window a system menu.

```
static const Style
  systemMenu;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

systemModal Sets the frame window in system-modal mode.

```
static const Style
  systemModal;
```

Win	PM	Motif
I	*Y*	*Y*

titleBar Gives the frame window a title bar.

```
static const Style
  titleBar;
```

Win	PM	Motif
Y	*Y*	*Y*

verticalScroll Gives the frame window a vertical scroll bar.

```
static const Style
  verticalScroll;
```

Win	PM	Motif
Y	*Y*	*Y*

windowList Adds an entry representing the frame window to the system window list.

```
static const Style
  windowList;
```

Win	PM	Motif
Y	*Y*	*Y*

Win This style has no effect with the Windows NT Program Manager shell. Top level windows are always added to the task list. With the Windows 95 shell, setting this style causes an entry to be placed into the task bar when the frame window is created.

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IFrameWindow contains the following nested classes:

IFrameWindow::Style (see page 392)

FrameSource

```
FrameSource {
    dialogResource,
    noDialogResource,
    tryDialogResource
    };
```

Use these enumerators to control the processing of a set of the IFrameWindow constructors (p. 357).

dialogResource
The constructors will attempt to load a dialog resource. If the dialog cannot be loaded, an exception is thrown.

noDialogResource
The constructors will create a new frame window with the style returned by IFrameWindow::defaultStyle (p. 370). No attempt is made to load a dialog resource.

tryDialogResource
The constructors will attempt to load a dialog resource. If the dialog resource cannot be found, a new frame window is created with the style returned by IFrameWindow::defaultStyle (p. 370).

Location

```
Location {
    leftOfTitleBar,  rightOfTitleBar, leftOfMenuBar,   rightOfMenuBar,
    leftOfClient,    rightOfClient,   aboveClient,     belowClient
    };
```

Use these enumerators to specify the possible locations for an extension:

leftOfTitleBar
Specifies left of the frame title area and, if a system menu is present, to the right of the menu.

© IBM Corp. 1992, 1996

rightOfTitleBar
Specifies right of the frame title.

leftOfMenuBar
Specifies left of the menu bar.

rightOfMenuBar
Specifies right of the menu bar.

leftOfClient
Specifies the left-hand side of the client area.

rightOfClient
Specifies the right-hand side of the client area.

aboveClient
Specifies the top of the client area.

belowClient
Specifies the bottom of the client area.

Win Windows does not support the following locations because it cannot position IFrameExtension objects to the side of the title bar or to the side of the menu.

- leftOfMenuBar
- leftOfTitleBar
- rightOfMenuBar
- rightOfTitleBar

Motif AIX does not support the following locations because it cannot position IFrameExtension objects to the side of the title bar or to the side of the menu.

- leftOfMenuBar
- leftOfTitleBar
- rightOfMenuBar
- rightOfTitleBar

SeparatorType

```
SeparatorType {
    none,
    thinLine,
    thickLine
    };
```

Use these enumerators to specify the possible types of separators drawn between the extension and the control it is attached to:

none
Draws no separator.

thinLine
Draws a thin separator line (1 pel wide).

thickLine
Draws a thick separator line (3 pels wide).

© IBM Corp. 1992, 1996

IFrameWindow::Style

Derivation

IBase
IBitFlag
IFrameWindow::Style

Inherited By None.

Header File iframe.hpp

The nested class IFrameWindow::Style represents properties of a frame window. The class for the IFrameWindow (p. 349) defines IFrameWindow::Style objects (p. 383) that you can use when constructing an IFrameWindow object.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

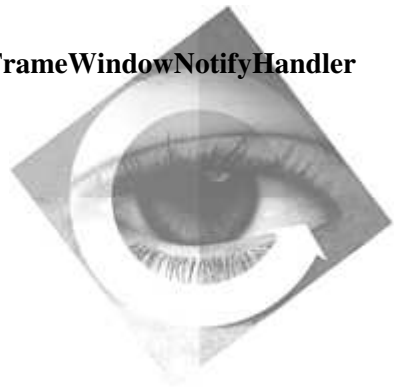
Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



IFrameWindowNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        IFrameWindowNotifyHandler

Inherited By None.

Header File iframnhd.hpp

Members

Member	Page
Constructor	393
dispatchHandlerEvent	394
~IFrameWindowNotifyHandler	394

The IFrameWindowNotifyHandler class processes events for all classes of frame windows.

This class is designed to handle events that require the frame window class to generate a notification. If notifications are enabled for the frame window, a notification will be generated and sent to all its observers when the proper conditions for the specific notification exist.

If you create a class derived from IFrameWindow (p. 349) that needs to notify observers of additional events, create a class derived from IFrameWindowNotifyHandler that overrides dispatchHandlerEvent (p. 394). In the class derived from IFrameWindow, implement enableNotification (p. 369) to call IWindow::setNotificationHandler (p. 1092), passing it an object of the class derived from IFrameWindowNotifyHandler.

Public Functions

Constructors

You can construct and destruct objects of this class.

IFrameWindowNotifyHandler

Provides the default constructor.

© Copyright IBM Corp. 1992, 1996

IFrameWindowNotifyHandler

Note: Generally, you do not need to construct an object of this class. Calling IFrameWindow::enableNotification (p. 369) causes an IFrameWindowNotifyHandler object to be constructed and added to the window, if necessary.

```
IFrameWindowNotifyHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

~IFrameWindowNotifyHandler

```
virtual
  ~IFrameWindowNotifyHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

Notification handlers process events that are sent or posted to a window by calling observer objects interested in those events.

dispatchHandlerEvent

Notifies the frame window observers when it receives any of the following events:

- Activate event
- Deactivate event

This function also calls IWindowNotifyHandler::dispatchHandlerEvent (p. 1112) so that observers will also be notified of generic window events.

If you create a class derived from IFrameWindowNotifyHandler, its dispatchHandlerEvent function should call IFrameWindowNotifyHandler::dispatchHandlerEvent for events it does not process.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IGroupBox

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IGroupBox

Inherited By None.

Header File igroupbx.hpp

Members

Member	Page	Member	Page
Constructor	397	position	400
calcMinimumSize	401	rect	400
classDefaultStyle	402	setDefaultStyle	399
convertToGUIStyle	399	setText	398
defaultStyle	399	show	400
foregroundColor	397	size	400
hide	400	visibleRectangle	398
moveSizeTo	400	~IGroupBox	398

The IGroupBox class creates and manages group box control windows. Group boxes are drawn around one or more associated controls. For example, you can group a set of related radio buttons within a group box in order to confine their mutually exclusive behavior. This allows the user to select a radio button outside the group box without deselecting one that is inside it.

A group box does not modify the behavior of any of the grouped controls. A group box does not accept input.

Note: To have a group box appear around the contents of a set canvas, use ISetCanvas::setText (Vol. III) instead of creating an IGroupBox object.

Motif The AIX release of the User Interface Class Library does not support the IGroupBox class.

Public Functions

© Copyright IBM Corp. 1992, 1996

Colors

Use these members to query colors for IGroupBox objects.

foregroundColor

Returns the foreground color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
I	*Y*	*N*

Win Returns the default foreground color value of the window area.

Constructors

You can construct and destruct objects of the IGroupBox class. You cannot copy or assign IGroupBox objects because both the copy constructor and the assignment operator are private functions.

IGroupBox

1
```
IGroupBox( unsigned long id,
           IWindow* parent,
           IWindow* owner,
           const IRectangle& initial = IRectangle ( ),
           const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	*Y*	*Y*

Use this constructor to construct a group box control and object from a control ID, parent and owner windows, a rectangle, and a style.

id Identifier of the group box control you construct.

parent The parent window of the group box control you construct. You must specify a parent window.

owner Owner window of the group box control you construct.

initial The initial position and size of the group box control you construct. Optional.

style Group box control's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

© IBM Corp. 1992, 1996

IGroupBox

2
```
IGroupBox( unsigned long id,
           IWindow* parentDialog );
```

Win	PM	Motif
Y	Y	Y

Use this constructor to create an object for the specified group box control from the ID of the group box control on a dialog window and the parent window.

id Identifier of the group box control you construct.
parent The parent window.

3
```
IGroupBox( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

Use this constructor to create an IGroupBox object from the window handle of an existing group box control.

handle The window handle of an existing group box control.

~IGroupBox

```
virtual
  ~IGroupBox();
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

setText

Sets the text for the group box and overrides the inherited setText. If the control's parent is a canvas, it is notified to update the layout for its children, if appropriate.

1
```
virtual IGroupBox&
  setText( const IResourceId& text );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the text by specifying an IResourceId to identify the string table ID and specific resource library.

2
```
virtual IGroupBox&
  setText( const char* text );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the group box text with a char *.

visibleRectangle

Returns a rectangle that represents the area of the window that is actually painted by the control, not the actual rectangle of the control.

```
virtual IRectangle
  visibleRectangle() const;
```

Win	PM	Motif
Y	Y	N

Styles

These style members provide a set of valid group box styles for this class. Use these members to query and set the group box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, will be returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 402) unless you have changed it using setDefaultStyle (p. 399).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent group boxes.

style Use the styles provided by IGroupBox Styles (p. 402) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

style Use the styles provided by IGroupBox (p. 402) Styles to specify the default style.

© IBM Corp. 1992, 1996

Window Positioning

Use these members to set and query the size and position of the group box.

hide Hides the group box.

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

Win	PM	Motif
N	N	Y

moveSizeTo Changes the position and size of the window, converting from the coordinates understood by this application to window system coordinates.

Note: OS/2 Presentation Manager coordinates are based on an origin in the bottom-left corner. Motif coordinates are based on an origin in the upper-left corner.

```
virtual IGroupBox&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	N	Y

position Returns the window's position converted to the coordinates understood by this application from window system coordinates (if they are different).

```
virtual IPoint
  position() const;
```

Win	PM	Motif
N	N	Y

rect Returns a rectangle representing the position and size of the window. If the application uses different coordinates than the window system coordinates, this function converts from window system coordinates to the coordinates understood by the application.

Note: This function returns IRectangle(0,0,0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual IRectangle
  rect() const;
```

Win	PM	Motif
N	N	Y

show Shows or hides the group box. If the flag is true the group box is shown. If the flag is false the group box is hidden.

```
virtual IWindow&
  show( IBase::Boolean );
```

Win	PM	Motif
N	N	Y

size Returns the size of the window in window coordinates.

Note: This function returns ISize(0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual ISize
  size() const;
```

Win	PM	Motif
N	*N*	*Y*

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Layout Support

Layout support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size of the group box control based on the current font and the text string.

© IBM Corp. 1992, 1996

IGroupBox

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid group box styles for this class. Use these members to query and set the group box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

classDefaultStyle

Provides the original default style for this class, which is the following: IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

ITextControl		
textId		

IWindow		
activeColorId	**disabledBackgroundColorId**	**noStyle**
backgroundColorId	**disabledForegroundColorId**	**positionId**
borderColorId	**enableId**	**saveBits**
clipChildren	**focusId**	**shadowColorId**

IWindow		
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IGroupBox contains the following nested classes:

IGroupBox::Style (see page 404)

© IBM Corp. 1992, 1996

IGroupBox::Style

Derivation

IBase
IBitFlag
IGroupBox::Style

Inherited By None.

Header File igroupbx.hpp

The nested class IGroupBox::Style provides a set of valid styles for the IGroupBox (p. 396) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IGUIColor

Derivation

IBase
  IVBase
    IColor
      IGUIColor

Inherited By None.

Header File icolor.hpp

Members

Member	Page	Member	Page
Constructor	406	systemColor	406
setColor	406	~IGUIColor	406

The IGUIColor class represents system-default colors for different areas of different types of windows. Like IColor (p. 174) objects created with the constructor that accepts a IColor::SystemColor (p. 181) enumeration value, the color returned by the object will represent the current system setting for the screen area, even if the system settings are changed after the IGUIColor object is created. When you use IGUIColor objects created with this constructor to set the color of an IWindow object, the window is updated automatically with the new color when the system settings are changed.

If you use the setBlue (p. 175), setRed (p. 175), or setGreen (p. 175) functions to modify the object, the link to the system color setting is broken.

PM IGUIColor objects correspond to one of the system SYSCLR_* colors in Presentation Manager.

Win IGUIColor objects correspond to one of the system COLOR_* colors in the Windows SDK.

Motif You cannot determine the color of an area for a widget that has not been created. You can query a widget for the color of all of the color areas defined for that widget. Using X, you can set the color of each widget through resource files. Because of this, it is impossible to determine the default color of an unknown widget.

In AIX, this class only provides hard-coded, default color values. If you want your application to be portable, use the class IColor (p. 174) to create IColor objects. Such objects give you control over color values.

© Copyright IBM Corp. 1992, 1996

Public Functions

Color Representation

Use these members to identify the system color represented by an IGUIColor object.

systemColor Returns the IColor::SystemColor (p. 181) enumerator used to create this object.

```
SystemColor
  systemColor() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

IGUIColor You can only construct objects of this class from a system color value identified by an IColor::SystemColor (p. 181) enumerator.

```
IGUIColor( SystemColor value );
```

Win	PM	Motif
Y	Y	Y

~IGUIColor

```
virtual
  ~IGUIColor();
```

Win	PM	Motif
Y	Y	Y

Setting System Colors

System colors define the default colors for the windows of all applications.

setColor Changes the system color represented by the object to the specified color. Changing a system color will affect the color of all windows, including those created by other applications, that use that system color.

```
IGUIColor&
  setColor( const IColor& newColor );
```

Win	PM	Motif
Y	Y	I

Exceptions	
IInvalidRequest	The object does not represent a valid system color, or the presentation system was unable to change the system color.

Inherited Public Functions

IColor		
asPixel	index	redMix
asRGBLong	operator !=	setBlue
blueMix	operator =	setGreen
greenMix	operator ==	setRed

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IHandle

Derivation

IBase
IHandle

Inherited By

IAccelTblHandle
IAnchorBlockHandle
IBitmapHandle
IContextHandle
IDisplayHandle
IEnumHandle
IMenuHandle
IMessageQueueHandle
IModuleHandle
IPageHandle
IPointerHandle
IPresSpaceHandle
IProcessId
IProfileHandle
IRegionHandle
ISemaphoreHandle
IStringHandle
IThreadHandle
IThreadId
IWindowHandle

Header File ibhandle.hpp

Members

Member	Page	Member	Page
Constructor	409	asUnsigned	409
asDebugInfo	409	handle	410
asString	409	operator Value	409

The IHandle class is a base class for all of the concrete handle-derived classes. You can create objects of this class, although such objects have limited utility. The User Interface Class Library APIs specify classes derived from IHandle.

The advantage of providing separate classes for each handle type is that they become distinct enough to prevent a user from passing the wrong handle type to a given function.

This base class manages the handle data member by defining the following:

- A constructor that accepts an unsigned long and stores the handle in the handle field.
- A user conversion operator as type IHandle. This operator permits IHandle objects to be used anywhere an IHandle is required (typically on windowing system or host operating system APIs).

There are numerous derived classes of IHandle representing each of the supported system handle types.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

You can construct objects of this class.

IHandle This constructor accepts an object of IHandle type Value as an argument. Typically this value originates in a system API call.

```
IHandle( Value value );
```

Win	PM	Motif
Y	*Y*	*Y*

Diagnostics

Use these members to get and set the accessible attributes of objects of this class.

asDebugInfo Returns the handle as a string containing diagnostic information.

```
IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	*Y*	*Y*

asString Returns the handle as a string of form *nnnn*.

```
IString
  asString() const;
```

Win	PM	Motif
Y	*Y*	*Y*

asUnsigned Returns the handle value as an unsigned long value.

```
unsigned long
  asUnsigned() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
Y	*Y*	*N*

© IBM Corp. 1992, 1996

IHandle

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Data

Value

These members contain information about this class.

handle This member is used to hold the handle value.

```
Value
  handle;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

Value

```
typedef unsigned long Value;
```

IHandler

Derivation

IBase
  IVBase
    IHandler

Inherited By

IClipboardHandler
ICnrDrawHandler
ICnrEditHandler
ICnrHandler
ICommandHandler
ICustomButtonDrawHandler
IDDEClientConversation
IDDETopicServer
IDMHandler
IEditHandler
IFileDialogHandler
IFlyOverHelpHandler
IFocusHandler
IFontDialogHandler
IFrameHandler
IHelpHandler
IKeyboardHandler
IListBoxDrawItemHandler
IMenuDrawItemHandler
IMenuHandler
IMMDeviceHandler
IMouseHandler
IPageHandler
IPaintHandler
IRecoordHandler
IResizeHandler
IScrollHandler
ISelectHandler
IShowListHandler
ISliderArmHandler
ISliderDrawHandler
ISpinHandler
IWindowNotifyHandler

Header File ihandler.hpp

Members

Member	Page	Member	Page
Constructor	412	enable	413
asDebugInfo	412	handleEventsFor	413
asString	413	isEnabled	413
defaultProcedure	415	stopHandlingEventsFor	414
disable	413	~IHandler	412
dispatchHandlerEvent	414		

The IHandler class is an abstract base class for window event handlers. You can write derived classes to handle specific window events, such as GUI window messages. Windows (or the objects that create windows) add objects of these handler classes to themselves by using IHandler::handleEventsFor (p. 413). Windows can also add handlers by using IWindow::addHandler (p. 1085).

You can add multiple handlers to a given window. The handlers are called in turn until one of them indicates that further handlers are not to be called. The handler achieves this by returning true from its dispatchHandlerEvent (p. 414).

© Copyright IBM Corp. 1992, 1996

IHandler

The handler that processes a given event is also responsible for setting the appropriate event result. The handler does this by using IEvent::setResult (p. 309). In general, a handler that returns true from dispatchHandlerEvent must call IEvent::setResult on the event passed to it. This result is returned to the sender of the event.

Handlers can be detached from windows by using either IHandler::stopHandlingEventsFor (p. 414) or IWindow::removeHandler (p. 1088). Ensure that you detach the handler from the windows it is handling prior to destroying the handler object. Typically, a window removes its handlers in its destructor.

Handlers can be disabled to suppress their handling of events without removing them from the window handler chain.

PM If none of the handlers returns true, the original window procedure for the underlying Presentation Manager window is called.

Motif If none of the handlers returns true, nothing happens.

Public Functions

Constructors

Only derived classes can call constructors of this class.

IHandler Derived classes call this default constructor from their constructors. This constructor initializes the handler in an enabled state.

```
IHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

~IHandler

```
virtual
  ~IHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Diagnostics

These members provide diagnostic information.

asDebugInfo Returns diagnostic information about the handler.

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString Returns the string "IHandler(enabled)" or "IHandler(disabled)".

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Enabling and Disabling Handlers

Use these members to enable and disable the handler.

disable Disables the handler so that it processes no window events.

```
virtual IHandler&
  disable();
```

Win	PM	Motif
Y	Y	Y

enable Enables the handler, by default. It accepts an optional *Boolean* parameter that, when false, causes the handler to be disabled.

```
virtual IHandler&
  enable( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

isEnabled Returns whether the handler is currently enabled.

```
Boolean
  isEnabled() const;
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects.

handleEventsFor

Attaches the handler to the specified IWindow (p. 1044) object. The handler's dispatchHandlerEvent (p. 414) function is called to process all events sent or posted to the window.

```
virtual IHandler&
  handleEventsFor( IWindow* window );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

stopHandlingEventsFor

Detaches the handler from the specified IWindow (p. 1044) object. The handler's dispatchHandlerEvent (p. 414) function is no longer called to process events sent or posted to the window.

```
virtual IHandler&
  stopHandlingEventsFor( IWindow* window );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects.

dispatchHandlerEvent

IWindow (p. 1044) calls this function when a window event occurs. 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 (p. 304) object.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event ) = 0;
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members provide the interface or implementation for handling an event.

defaultProcedure

Sends the specified IEvent (p. 304) object to the original window procedure for the underlying window.

```
virtual IEventResult
  defaultProcedure( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Motif AIX does not support this function.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IHelpErrorEvent

Derivation

IBase
IVBase
IEvent
IHelpErrorEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page
Constructor	416
error	417
~IHelpErrorEvent	416

Events of class IHelpErrorEvent are dispatched when a help-related error occurs.

Public Functions

Constructors

You can construct and destruct objects of this class.

IHelpErrorEvent

Constructs an IHelpErrorEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function IHelpHandler::handleError (p. 424).

```
IHelpErrorEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IHelpErrorEvent

```
virtual
  ~IHelpErrorEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Error Type

IHelpErrorEvent objects contain a value that represents a help-related error.

error Returns the type of help-related error. The returned value is an ErrorType (p. 417) enumerator.

```
ErrorType
  error() const;
```

Win	PM	Motif
Y	*Y*	*Y*

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**	

ErrorType

```
ErrorType {
    loadingDLL,           noWindow,           invalidAppHandle,
    invalidInstanceHandle, noHelpInstance,    invalidQueryHandle,
    noHelpTable,          invalidId,          noIndex,
    noContent,            openHelpFile,       readHelpFile,
    closeHelpFile,        invalidHelpFile,    notEnoughMemory,
    unableFreeMemory,     invalidHelpWindow,  unopenDatabase,
```

© IBM Corp. 1992, 1996

```
    fontNotFound,          unknown
    };
```

These enumerators specify the type of error that occurred:

loadingDLL
Specifies that the application is unable to load the dynamic link library (DLL) containing the help table resource.

noWindow
Specifies that the window to be associated with the help window object is not a frame window and has no frame window in its parent window chain.

invalidAppHandle
Specifies that the application window to be associated with the help window object does not have a valid window handle.

invalidInstanceHandle
Specifies that the help window object to be associated with the application window does not have a valid window handle.

noHelpInstance
Specifies that neither the frame windows in the parent chain of the specified application window nor its owner window has an associated help window object.

invalidQueryHandle
Specifies that the application window used to query the help window object does not have a valid window handle.

noHelpTable
Specifies that the application did not provide a help table for contextual help.

invalidId Specifies that the ID of the main help item specified for a contextual help request was not found in the help table.

noIndex Specifies that the help library file does not contain an index.

noContent Specifies that the help library file does not have any content.

openHelpFile
Specifies that the help library file cannot be opened.

readHelpFile
Specifies that the help library file cannot be read.

closeHelpFile
Specifies that the help library file cannot be closed.

invalidHelpFile
Specifies that an improper help library file was provided.

notEnoughMemory
Specifies that Information Presentation Facility (IPF) is unable to allocate the memory it needs.

unableFreeMemory
Specifies that IPF is unable to free memory it allocated.

invalidHelpWindow
Specifies that IPF is unable to find the requested help window in the specified help library files.

unopenDatabase
Specifies that IPF is unable to read the unopened help library file.

fontNotFound
Specifies that IPF cannot load a font it requires.

unknown Specifies that another error occurred.

© IBM Corp. 1992, 1996

IHelpHandler

Derivation

IBase
IVBase
IHandler
IHelpHandler

Inherited By None.

Header File ihelphdr.hpp

Members

Member	Page	Member	Page
Constructor	422	showCoverPage	426
controlSelect	424	showHistory	426
dispatchHandlerEvent	424	showIndex	426
handleError	424	showPage	426
handleEventsFor	423	showSearchList	426
helpUndefined	424	showTutorial	427
hyperlinkSelect	425	stopHandlingEventsFor	423
keysHelpId	425	subitemNotFound	427
menuBarCommand	425	swapPage	427
openLibrary	425	~IHelpHandler	422
showContents	425		

The IHelpHandler class processes events related to a help window. While you can provide help support without using the IHelpHandler class, and can provide basic error support without deriving from IHelpHandler, you need to create a class derived from IHelpHandler to provide more advanced help support.

You attach a help handler object to an application frame window by calling IHelpHandler::handleEventsFor (p. 423), passing the window to the help handler. Only attach help handlers to frame windows that you have associated with a help window, either when constructing an IHelpWindow (p. 442) object or when calling IHelpWindow::setAssociatedWindow (p. 444).

A help handler allows you to process the following help events:

- Requests for help when the following occurs:
  - The user selects the **Keys Help** menu choice.

    You can process this request by overriding the function keysHelpId (p. 425).
  - The active control has no contextual help panel defined in the help table resource.

© Copyright IBM Corp. 1992, 1996

You can dynamically display a contextual help panel for such a control by overriding the function subitemNotFound (p. 427).

  - The active frame window has no general help defined.

    You can process this request by overriding the function helpUndefined (p. 424).

- Errors related to help processing. You can process these errors by overriding the function handleError (p. 424).
- Selection of the **Tutorial** menu choice. You can process this action by overriding the function showTutorial (p. 427).
- Notification of the following:
  - The help cover page window is about to be shown.

    To process, override the function showCoverPage (p. 426).

  - A page within the cover page window is about to be shown.

    To process, override the function showPage (p. 426).

  - A page is about to be swapped.

    To process, override the function swapPage (p. 427).

  - The help Contents window is about to be shown.

    To process, override the function showContents (p. 425).

  - The Viewed Pages window is about to be shown.

    To process, override the function showHistory (p. 426).

  - The help Index window is about to be shown.

    To process, override the function showIndex (p. 426).

  - The search list window is about to be shown.

    To process, override the function showSearchList (p. 426).

  - The Libraries list window is about to be shown.

    To process, override the function openLibrary (p. 425).

  - The user selected an application-added control on a help window.

    To process, override the function controlSelect (p. 424).

- Selection of an application-added item on a customized help menu bar. You can process this action by overriding the function menuBarCommand (p. 425).
- Selection of a special link on a help panel. You can process this action by overriding the function hyperlinkSelect (p. 425).

© IBM Corp. 1992, 1996

When the help handler receives one of these help events, it creates a corresponding event object and routes that object to the appropriate IHelpHandler virtual function. Override these virtual functions to supply your own specialized processing of these events.

The return value from the virtual functions specifies whether the event is passed on for additional processing, as follows:

true The help event requires no additional processing. Do not pass it to another handler.

false Pass the help event to the next handler for additional processing, as follows:

- If there is another handler for the application frame window, pass the event to the next handler.
- If this is the last handler for the application frame window, call IWindow::defaultProcedure (p. 1082) to process the help event.

Public Functions

Constructors

You can construct and destruct objects of this class. You can construct IHelpHandler objects using its default constructor, which takes no parameters.

IHelpHandler

```
IHelpHandler();
```

Win	PM	Motif
Y	Y	Y

~IHelpHandler

```
virtual
  ~IHelpHandler();
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to each of the handlers attached to that window. It does this by calling the event-dispatching function of each handler object.

An IHelpHandler object processes only help-related events and can only be attached to and detached from a frame window.

handleEventsFor

Attaches the handler to a frame window. The application frame window must be one of the following:

- The window you specified when constructing an IHelpWindow (p. 442) object (this is typically a primary window)
- A window you specified when calling the function IHelpWindow::setAssociatedWindow (p. 444)

```
virtual IHelpHandler&
  handleEventsFor( IFrameWindow* associatedWindow );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	*associatedWindow* must be nonzero.

stopHandlingEventsFor

Detaches the handler from the application frame window.

```
virtual IHelpHandler&
  stopHandlingEventsFor( IFrameWindow* associatedWindow );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	*associatedWindow* must be nonzero.

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to each of the handlers attached to that window. It does this by calling the event-dispatching function of each handler object.

An IHelpHandler object processes only help-related events and can only be attached to and detached from a frame window.

dispatchHandlerEvent

If this handler receives a help event, this function calls the appropriate virtual function to process it.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

A help handler contains event-processing members that you can use to provide application-specific processing for help-related events.

controlSelect Called when the user selects an application-added control on a help window. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  controlSelect( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

handleError Called when a help-related error occurs. By default, this function displays a message box indicating the error type of the IHelpErrorEvent (p. 416). The error type is an enumerator provided by ErrorType (p. 417).

```
virtual Boolean
  handleError( IHelpErrorEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

helpUndefined

Called when the user requests general help and it is not defined. By default, this function does nothing, setting the event result to false. Your application can override this function and do either of the following:

- Display its own help panel
- Use IHelpWindow::show (p. 449) to display a particular help panel or window.

```
virtual Boolean
  helpUndefined( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

hyperlinkSelect

Called when the user refers to a heading that contains an inform link. This informs the application that the heading is accessed. You specify this link using the *reftype=inform* attribute on the Information Presentation Facility (IPF) *link* tag. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  hyperlinkSelect( IHelpHyperlinkEvent& event );
```

Win	PM	Motif
Y	Y	Y

keysHelpId

Called when the user selects the **Keys Help** choice from a help window or from a menu. By default, this function sets the event result to 0, indicating that no panel is to be displayed. You can override the default action by setting the event result to the ID of the help panel that you want IPF to display as the keys help panel.

```
virtual Boolean
  keysHelpId( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

menuBarCommand

Called when the user selects an application-added item on a customized menu bar used by the help window. By default, this function does nothing, setting the event result to false.

Use the function IHelpWindow::Settings::setMenuBar (p. 462) to assign a custom menu bar to the help window.

```
virtual Boolean
  menuBarCommand( IHelpMenuBarEvent& event );
```

Win	PM	Motif
Y	Y	Y

openLibrary

Called when the Libraries list window is about to be displayed. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  openLibrary( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showContents

Called when the help Contents window is about to be displayed. By default, this function does nothing, setting the event result to false.

© IBM Corp. 1992, 1996

```
virtual Boolean
  showContents( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showCoverPage

Called when the help cover page window is about to be displayed. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showCoverPage( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showHistory

Called when the Viewed Pages window is about to be displayed. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showHistory( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showIndex

Called when the help Index window is about to be displayed. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showIndex( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showPage

Called when a child window of the cover page is about to be opened. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showPage( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	Y	I

Motif IPF/X does not generate an event for this condition.

showSearchList

Called when the search list window is about to be displayed. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showSearchList( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	*Y*	*I*

Motif IPF/X does not generate an event for this condition.

showTutorial Called when the user selects the **Tutorial** choice from a help window or from a menu. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  showTutorial( IHelpTutorialEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

subitemNotFound

Called when the user requests help on a control or window that cannot be found in the help table that is currently in use for the IHelpWindow object. By default, this function does nothing but return false, resulting in extended help being shown.

You can provide your own functionality for this event by overriding this function. To prevent extended help from being shown, set the event result to true using IEvent::setResult (p. 309) on the IHelpSubitemNotFoundEvent (p. 437) object passed into the function.

Returning true from the function itself will prevent other handlers from processing this event.

```
virtual Boolean
  subitemNotFound( IHelpSubitemNotFoundEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

swapPage Called when a child window of the cover page is about to be swapped. By default, this function does nothing, setting the event result to false.

```
virtual Boolean
  swapPage( IHelpNotifyEvent& event );
```

Win	PM	Motif
Y	*Y*	*I*

Motif IPF/X does not generate an event for this condition.

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

© IBM Corp. 1992, 1996

IHelpHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IHelpHyperlinkEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpHyperlinkEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page
Constructor	429
id	430
~IHelpHyperlinkEvent	430

Events of class IHelpHyperlinkEvent are dispatched when the user refers to a heading that contains an inform link. This informs the application that the heading is accessed.

You specify an inform link using the *reftype=inform* attribute on the Information Presentation Facility (IPF) *:link* tag. For example, the following link informs the application when the heading that contains it is referred to:

```
:h1 res=123.Title:elink.
:link auto reftype=inform res=1000.
```

The resource value (*res=1000*) identifies the link to the application and directs some application-specific function.

Do not confuse this link type with common hypertext links such as the following:

```
:link reftype=hd res=123.Title:elink.
```

Public Functions

Constructors

You can construct and destruct objects of this class.

IHelpHyperlinkEvent

Constructs an IHelpHyperlinkEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an

© Copyright IBM Corp. 1992, 1996

IHelpHyperlinkEvent

object of the class IEvent (p. 304) and passes the resulting objects to the function IHelpHandler::hyperlinkSelect (p. 425).

```
IHelpHyperlinkEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IHelpHyperlinkEvent

```
virtual
  ~IHelpHyperlinkEvent();
```

Win	PM	Motif
Y	Y	Y

Link Information

An IHelpHyperlinkEvent object contains information that identifies the inform link selected by the user.

id

Returns the resource ID of the inform link. Your application uses this identifier to determine application-specific processing for the link.

```
unsigned long
  id() const;
```

Win	PM	Motif
Y	Y	Y

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	

IHelpMenuBarEvent

Derivation

IBase
IVBase
IEvent
IHelpMenuBarEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page
Constructor	432
commandId	433
~IHelpMenuBarEvent	432

Objects of the IHelpMenuBarEvent class are dispatched when the user selects an application-added item from a customized menu bar on a help window.

Public Functions

Constructors

You can construct and destruct objects of this class.

IHelpMenuBarEvent

Construct an IHelpMenuBarEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting objects to the function IHelpHandler::menuBarCommand (p. 425).

```
IHelpMenuBarEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IHelpMenuBarEvent

```
virtual
  ~IHelpMenuBarEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Menu Information

An IHelpMenuBarEvent object contains information that identifies the application-specific menu item selected by the user from the menu bar of a help window.

commandId Returns the ID of the menu bar item selected.

```
unsigned long
  commandId() const;
```

Win	PM	Motif
Y	*Y*	*Y*

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**	

© IBM Corp. 1992, 1996

IHelpNotifyEvent

Derivation

IBase
IVBase
IEvent
IHelpNotifyEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page	Member	Page
Constructor	435	windowHandle	435
controlId	435	~IHelpNotifyEvent	435

The IHelpNotifyEvent class events are dispatched when the user initiates a help-related action that your associated application window can control. Your application can then change the behavior or appearance of the appropriate help window.

An IHelpNotifyEvent represents one of the following actions:

- Showing the help cover page window
- Showing a page within the cover page window
- Swapping of a page
- Showing the help Contents window
- Showing the Viewed Pages window
- Showing the help Index window
- Showing the search list window
- Showing the help Libraries window
- Selecting of a control added by an application on a help window

The IHelpNotifyEvent object identifies the event type and the associated help window or selected control.

Public Functions

Constructors

You can construct and destruct objects of this class.

© Copyright IBM Corp. 1992, 1996

IHelpNotifyEvent

Constructs an IHelpNotifyEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an object of the class IEvent (p. 304).

```
IHelpNotifyEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IHelpNotifyEvent

```
virtual
  ~IHelpNotifyEvent();
```

Win	PM	Motif
Y	Y	Y

Help Window

An IHelpNotifyEvent object contains information that identifies a help window associated with a particular help-related notification.

windowHandle

Returns the window handle of either the help window being shown or swapped or the help window containing the control that the user selected.

```
IWindowHandle
  windowHandle() const;
```

Win	PM	Motif
Y	Y	Y

Selected Control

An IHelpNotifyEvent object can identify a control when it is added to a help window by the application and selected by the user.

controlId

Returns the window identifier of the help control that the user selected. For example, this might be the window identifier of a push button you added to a help window. If the event does not represent the selection of a control, this function returns 0.

```
unsigned long
  controlId() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

IHelpNotifyEvent

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**	

IHelpSubitemNotFoundEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpSubitemNotFoundEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page	Member	Page
Constructor	437	subtopicId	438
isFrame	438	topicId	438
isMenu	438	~IHelpSubitemNotFoundEvent	437
isWindow	438		

Events of class IHelpSubitemNotFoundEvent are dispatched when the user requests help on a control for which Information Presentation Facility (IPF) cannot find an entry in the help subtable (and thus, a control for which IPF cannot display a contextual help panel).

Public Functions

Constructors

You can construct and destruct objects of this class.

IHelpSubitemNotFoundEvent

Constructs an IHelpSubitemNotFoundEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting objects to the function IHelpHandler::subitemNotFound (p. 427).

```
IHelpSubitemNotFoundEvent( const IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

~IHelpSubitemNotFoundEvent

© Copyright IBM Corp. 1992, 1996

IHelpSubitemNotFoundEvent

```
virtual
  ~IHelpSubitemNotFoundEvent();
```

Win	PM	Motif
Y	Y	Y

Help Item

An IHelpSubitemNotFoundEvent object contains information that identifies the window for which the user requested help.

isFrame Queries whether help was requested on a frame window.

```
Boolean
  isFrame() const;
```

Win	PM	Motif
Y	Y	Y

isMenu Queries whether help was requested on a menu window.

```
Boolean
  isMenu() const;
```

Win	PM	Motif
Y	Y	Y

isWindow Queries whether help was requested on an application window.

```
Boolean
  isWindow() const;
```

Win	PM	Motif
Y	Y	Y

subtopicId If help was requested on an application or frame window, this function returns the identifier of the control with the input focus. If help was requested for a menu item, its identifier is returned.

```
unsigned long
  subtopicId() const;
```

Win	PM	Motif
Y	Y	Y

topicId Returns the identifier of the window or menu that contains the control or menu item that had the input focus when help was requested.

```
unsigned long
  topicId() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType

IEvent		
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**	

© IBM Corp. 1992, 1996

IHelpTutorialEvent

Derivation

IBase
IVBase
IEvent
IHelpTutorialEvent

Inherited By None.

Header File ihelpevt.hpp

Members

Member	Page
Constructor	440
tutorialName	441
~IHelpTutorialEvent	440

Events of class IHelpTutorialEvent are dispatched when the user selects the **Tutorial** menu choice.

Public Functions

Constructors

You can construct and destruct objects of this class.

IHelpTutorialEvent

Constructs an IHelpTutorialEvent using the specified event. IHelpHandler::dispatchHandlerEvent (p. 424) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting objects to the function IHelpHandler::showTutorial (p. 427).

```
IHelpTutorialEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IHelpTutorialEvent

```
virtual
  ~IHelpTutorialEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Tutorial

An IHelpTutorialEvent object contains the name of the tutorial program to run.

tutorialName Returns the tutorial program name. You specify this program when creating the IHelpWindow (p. 442) object.

```
IString
  tutorialName() const;
```

Win	PM	Motif
Y	*Y*	*Y*

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**	

© IBM Corp. 1992, 1996

IHelpWindow

Derivation

IBase
IVBase
INotifier
IWindow
IHelpWindow

Inherited By None.

Header File ihelp.hpp

Members

Member	Page	Member	Page
Constructor	445	moveTo	455
addLibraries	448	noStyle	457
classDefaultStyle	456	searchListWindow	454
communicationWindow	453	sendEvent	451
contentsWindow	453	setActiveWindow	444
coverPageWindow	453	setAssociatedWindow	444
defaultStyle	451	setDefaultStyle	451
handle	453	setHelpTable	449
helpWindow	453	setTitle	452
hide	449	setUsingHelp	449
hidePanelIds	450	show	449
indexWindow	454	showPanelIds	450
ipfCompatible	456	sizeTo	455
isIPFCompatible	451	viewedPagesWindow	454
moveSizeTo	454	~IHelpWindow	448

The IHelpWindow class provides help for application windows using Information Presentation Facility (IPF) and for native help on Windows. Typically, you create an IHelpWindow object and associate it with the primary windows of your application. Multiple IPF help windows can also exist within an application. When an application window is associated with a help window, help events are dispatched to the help handler attached to the associated application window.

Win By default, IHelpWindow objects constructed in the Windows environment use native Windows help. To prepare your program to use this style of help, you build a .hlp file using the Windows help compiler from a source file written in Rich Text Format (RTF) and then access the help file using the portable IHelpWindow classes. This approach gives you the ability to port your application code while providing your users with native look-and-feel when using help. However, you should note the following differences in IHelpWindow behavior.

© Copyright IBM Corp. 1992, 1996

- Native Windows help does not send messages to the application.
- Native Windows help does not position itself relative to the associated window.
- Because there is only one native help window instance per system, the help instance is only closed when all windows that have called it are closed.

If you prefer to show help using IPF on Windows, set the default IHelpWindow style before constructing the IHelpWindow object. You do this using the static IHelpWindow function:

```
IHelpWindow::setDefaultStyle (p. 451) (IHelpWindow::ipfCompatible (p. 456) );
```

You can also set the help window style by specifying the appropriate style on the IHelpWindow constructor.

You can build the viewable help files for your AIX application using the Information Presentation Facility compiler (IPFC) that comes with the Developer's Toolkit for OS/2, version 2.1 or later.

Instead of compiling an IPF file using code page 437, as you would to view help on the OS/2 platform, you must compile the file on the OS/2 platform using code page 850 to view the help on AIX.

IHelpWindow uses the window identifier of the IWindow object as the identifier of the contextual or general help panel to display.

The class default style for the IHelpWindow class is different in the Windows environment. In the OS/2 and AIX environments, the default is set to IHelpWindow::ipfCompatible (p. 456) and cannot be changed. In the Windows environment, the default is set for native Windows help. You can change the default to IHelpWindow::ipfCompatible (p. 456) style by calling IHelpWindow::setDefaultStyle (p. 451).

When using the native Windows help, many of the IHelpWindow methods, for example IHelpWindow::hidePanelIds (p. 450), perform no other function than to return a reference to the IHelpWindow object or null. Also, because native Windows help does not send messages back to the application, most of the IHelpHandler functions are never called. For more details, see the function descriptions for the IHelpWindow and IHelpHandler classes.

Public Functions

Application Windows

You can use the same IHelpWindow object to service help requests for many application windows created in the same thread, regardless of whether they are primary, secondary, or child

© IBM Corp. 1992, 1996

frame windows. You can control the help subtable that IPF uses to display contextual help by managing the *active* window, control where IPF positions a help panel by managing the *relative* window, and control the owner window of a help panel by managing the *associated* window.

setActiveWindow

Sets the application window from which IPF expects subsequent requests for contextual or general help. With this function, you can enable IPF to display contextual and general help panels for a child frame window using help tables and subtables. For this case, IPF would otherwise treat the parent frame window as the active window.

Optionally, you can specify the window to position the help window next to.

```
virtual IHelpWindow&
  setActiveWindow( IFrameWindow* activeWindow,
                   IFrameWindow* relativeWindow = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

activeWindow

Frame window that IPF treats as the active window when processing subsequent help requests.

IPF searches the help subtable for this frame to determine the help panel that determines the correct contextual help panel to display. For general help, it uses the general help panel as identified for this frame window in the help table.

Specify 0 to indicate that IPF should use its default algorithm for determining the active window.

relativeWindow

Frame window that IPF minimally overlaps with the help window.

Specify 0 to use the associated window.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), you can still use this function to control which help subtable to use when displaying contextual or general help. However, since there is only one native Windows help instance per system, the help window is not positioned relative to your application's window. Therefore, for native Windows help, setting the second parameter will have no effect.

setAssociatedWindow

Associates an application frame window with the help window. More than one window can be associated with a help window.

For you to provide an application window with contextual or general help using help tables, the window must be an associated window, have an associated window in its parent window chain, or be a secondary window owned by an associated window. IPF also sends help notification events to associated frame windows, positions the help window to minimally overlap an associated frame window, closes the help window when the associated frame window is closed, and activates the associated frame window when the user dismisses the help window.

This function allows you to use the same help window for multiple primary windows. Additionally, you can better manage help windows for a secondary frame window. This might be necessary if a primary window would otherwise be the associated window. By making a secondary frame window the associated window you get these results:

- IPF positions the help window relative to the secondary frame window.
- The help window closes when the secondary frame window closes.
- The secondary frame window becomes activated when the user dismisses the help window.

A frame window is automatically disassociated from a help window when it closes.

```
virtual IHelpWindow&
  setAssociatedWindow( IFrameWindow* associatedWindow );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	*associatedWindow* is 0. It must be a valid window handle.
IAccessError	Information Presentation Facility (IPF) could not associate the specified window with the help window. Possibly *associatedWindow* does not represent a valid window.

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IHelpWindow objects because both the copy constructor and assignment operator are private functions.

IHelpWindow

1
```
IHelpWindow(
   IFrameWindow* associatedWindow = 0,
   const IHelpWindow::Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

associatedWindow

Frame window to be associated with the help window.

For you to provide an application window with contextual or general help

© IBM Corp. 1992, 1996

using help tables, the window must be an associated window, have an associated window in its parent window chain, or be a secondary window owned by an associated window. IPF also sends help notification events to associated frame windows, positions the help window to minimally overlap an associated frame window, closes the help window when the associated frame window is closed, and activates the associated frame window when the user dismisses the help window.

If you do not specify an application frame window to be associated with the help window object, you can later call the function IHelpWindow::setAssociatedWindow (p. 444) to make this association.

style Type of help window to construct. If you do not specify *style*, the constructor calls IHelpWindow::defaultStyle (p. 451) to return the default style.

Exceptions	
IAccessError	Information Presentation Facility (IPF) could not create the help window.

2

```
IHelpWindow(
   const IResourceId& helpTable,
   IFrameWindow* associatedWindow,
   const IHelpWindow::Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

helpTable Resource identifier for a help table resource.

associatedWindow
Frame window to be associated with the help window.

For you to provide an application window with contextual or general help using help tables, the window must be an associated window, have an associated window in its parent window chain, or be a secondary window owned by an associated window. IPF also sends help notification events to associated frame windows, positions the help window to minimally overlap an associated frame window, closes the help window when the associated frame window is closed, and activates the associated frame window when the user dismisses the help window.

style Type of help window to construct. If you do not specify *style*, the constructor calls IHelpWindow::defaultStyle (p. 451) to return the default style.

3

```
IHelpWindow(
   const Settings& settings,
   IFrameWindow* associatedWindow = 0,
   const IHelpWindow::Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

settings An object of the class IHelpWindow::Settings (p. 459) to associate the help window with initial information, such as the following:

- Customized menu bar
- Accelerator table
- Title text
- Tutorial program
- Help panel libraries

associatedWindow
Frame window to be associated with the help window.

For you to provide an application window with contextual or general help using help tables, the window must be an associated window, have an associated window in its parent window chain, or be a secondary window owned by an associated window. IPF also sends help notification events to associated frame windows, positions the help window to minimally overlap an associated frame window, closes the help window when the associated frame window is closed, and activates the associated frame window when the user dismisses the help window.

You can also call the function IHelpWindow::setAssociatedWindow (p. 444) to associate a frame window with the IHelpWindow object.

style Type of help window to construct. If you do not specify *style*, the constructor calls IHelpWindow::defaultStyle (p. 451) to return the default style.

This is the most flexible IHelpWindow constructor because the IHelpWindow::Settings class allows you to specify a wide amount of information, including a tutorial, menu bar, and accelerator table, for which the IHelpWindow class does not provide member functions.

Win See IHelpWindow::Settings (p. 459) Windows notes for a description of which settings are supported in the Windows environment.

Exceptions	
IAccessError	Information Presentation Facility (IPF) could not create the help window.

4 `IHelpWindow( const IWindowHandle& helpWindowHandle );`

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

IHelpWindow

helpWindowHandle
Window handle of an existing help window.

Only this constructor allows you to create an IHelpWindow object for an existing help window. All other constructors create a help window.

Exceptions	
IInvalidParameter	*helpWindowHandle* is 0. It must be a valid window handle.

~IHelpWindow

```
virtual
  ~IHelpWindow();
```

Win	PM	Motif
Y	Y	Y

Help Panels

Most of the help information that a user sees is typically in the form of help panels.You can direct IPF to display help panels for contextual and general help based on a help table. You can also show specialized help panels or help windows programmatically, independent of help tables.

addLibraries

Adds a library or list of libraries to those already used by IPF. IPF searches these help libraries for the help panels that it displays. This function deals with binary help files that have an .hlp extension.

```
virtual IHelpWindow&
  addLibraries( const char* helpLibraryNames );
```

Win	PM	Motif
Y	Y	Y

helpLibraryNames
The list of help library names.

If you specify more than one library name, separate the names with a space character. If you specify a null string, IPF no longer references any library names that were previously specified using this function.

PM If you want to search the HELP path for the help library files, specify the file names without a directory or path.

Win When using the native Windows help style for IHelpWindow objects, you should only pass a single file name on the addLibraries call. The filename should identify the help file that was compiled with the Windows help compiler from a Rich Text Format (RTF) source file.

Exceptions	
IInvalidRequest	Information Presentation Facility (IPF) is unable to add the specified help libraries. Possibly it could not open the help library files.

hide Closes the help window.

```
virtual IHelpWindow&
  hide();
```

Win	PM	Motif
Y	Y	Y

setHelpTable Sets the help table for IPF to use for satisfying user requests for contextual and general help. Entries in the help table identify the help panel to be searched for in the help libraries, for a given application frame window or control.

This function replaces any help table identified with the class IHelpWindow::Settings (p. 459).

You can also associate help panels with IWindow or IMenu item objects directly by calling IWindow::setHelpId (p. 1064) or IMenu::setItemHelpId (p. 540).

```
virtual IHelpWindow&
  setHelpTable( const IResourceId& helpTable );
```

Win	PM	Motif
Y	Y	Y

helpTable Resource identifier for a help table resource.

setUsingHelp Sets the help panel to use instead of the default Using Help panel provided by IPF. Calling IHelpWindow::show (p. 449) and specifying IHelpWindow::usingHelp (p. 457) causes IPF to show this help panel.

This function replaces any Using Help panel identified with the class IHelpWindow::Settings (p. 459).

```
virtual IHelpWindow&
  setUsingHelp( unsigned long panelId );
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function will change the help panel that is displayed when calling IHelpWindow::show (p. 449) with the HelpType of usingHelp.

show Displays a help panel or specialized help window.

1

```
virtual IHelpWindow&
  show( const char* panelName );
```

Win	PM	Motif
Y	Y	Y

panelName
The name of the help panel.

Win If you are using native Windows help (that is, you did not construct your

© IBM Corp. 1992, 1996

IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function calls Windows help, using the panel name as a keyword.

2
```
virtual IHelpWindow&
  show( HelpType helpType );
```

Win	PM	Motif
Y	Y	Y

helpType Enumerator value from the HelpType (p. 457) enumeration.

The enumerator identifies a help panel or kind of help window.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), the HelpType (p. 457) enumeration values have different meanings.

3
```
virtual IHelpWindow&
  show( const IResourceId& panelId );
```

Win	PM	Motif
Y	Y	Y

panelId Identifier for the help panel from your help (.hlp) file that you want to show.

If you specify an identifier of 0, the Using Help window is displayed.

Note: The IResourceLibrary portion of the *panelId* value is ignored.

Panel Identifiers

You can uniquely identify a help panel by its help panel identifier. You can have help panels display their panel IDs for diagnostic purposes.

hidePanelIds

Removes the identifier for a help panel from its title bar text.

```
virtual IHelpWindow&
  hidePanelIds();
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

showPanelIds

Adds the identifier for a help panel to its title bar text.

```
virtual IHelpWindow&
  showPanelIds( Boolean visibleId = true );
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Sending and Posting Events

Event send and post members are overridden to ensure events get routed to the correct window.

sendEvent Sends the specified event to the help window.

1
```
virtual IEventResult
  sendEvent( EventType eventType,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

2
```
virtual IEventResult
  sendEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

3
```
virtual IEventResult
  sendEvent( const IEvent& event ) const;
```

Win	PM	Motif
Y	Y	Y

Styles

IHelpWindow defines objects of the nested class IHelpWindow::Style (p. 464). You can use these styles with IHelpWindow::setDefaultStyle (p. 451).

defaultStyle Returns the default style. This style is equivalent to IHelpWindow::classDefaultStyle (p. 456) unless you have changed the style using IHelpWindow::setDefaultStyle (p. 451).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

isIPFCompatible

Returns true if the IPF is being used to display application help information. True is always returned, except on Windows where true means IPF is being used and false means that native Windows help is being used.

```
Boolean
  isIPFCompatible() const;
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for construction of all subsequent help windows. You only need this function in the Windows environment and then only if you want to change which help manager you want to use to process application help requests.

© IBM Corp. 1992, 1996

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Win To change from native Windows help to IPF help, use:

```
setDefaultStyle(IHelpWindow::ipfCompatible (p. 456) ).

To change from IPF help to native Windows help, use:
setDefaultStyle(~IHelpWindow::ipfCompatible).
```

Title Text

The IHelpWindow class allows you to control the text displayed in the title bar of the cover page window.

setTitle Sets the title bar text for the help window.

These functions replace any title identified with the class IHelpWindow::Settings (p. 459).

1

```
virtual IHelpWindow&
  setTitle( const IResourceId& titleId );
```

Win	PM	Motif
Y	Y	Y

titleId Resource identifier of a string in a string table.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

2

```
virtual IHelpWindow&
  setTitle( const char* titleText );
```

Win	PM	Motif
Y	Y	Y

titleText Address of a character string.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Window Accessing

Many different help-related windows can be displayed during the course of an application. You can access the different types of windows to provide application-specific processing or customization.

communicationWindow

Returns the handle of the active communication window.

```
IWindowHandle
  communicationWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

contentsWindow

Returns the handle of the Table of Contents window.

```
IWindowHandle
  contentsWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

coverPageWindow

Returns the handle of the IPF multiple-document-interface parent window. This is the window within which all other IPF windows are displayed.

```
IWindowHandle
  coverPageWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

handle

Returns the handle of the help window.

```
virtual IWindowHandle
  handle() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), a unique window handle is returned, but since native Windows help provides only one help instance per system, the handle provides no other purpose than unique identification.

helpWindow

Returns the help window that is associated with the specified application window. The parent window chain is searched until an application window associated with a help window is found, in which case this function returns that help window. If a help window cannot be found, 0 is returned.

© IBM Corp. 1992, 1996

```
static IHelpWindow*
  helpWindow( const IWindow* window );
```

Win	PM	Motif
Y	Y	Y

indexWindow

Returns the handle of the Index window.

```
IWindowHandle
  indexWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

searchListWindow

Returns the handle of the Search window.

```
IWindowHandle
  searchListWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

viewedPagesWindow

Returns the handle of the Viewed Pages window.

```
IWindowHandle
  viewedPagesWindow() const;
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Window Positioning

The IHelpWindow class allows you to size and position the cover page window, within which all other IPF windows are displayed.

moveSizeTo

Changes the size and position of the help cover page window, within which all other IPF windows are displayed.

```
virtual IHelpWindow&
  moveSizeTo( const IRectangle& rectangle );
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function will move and size the single system instance of native Windows help.

moveTo Changes the position of the help cover page window, within which all other IPF windows are displayed. Call this function only when the help window is visible.

```
virtual IHelpWindow&
  moveTo( const IPoint& point );
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect. You can use IHelpWindow::moveSizeTo (p. 454) to change both the size and position of the single native Windows help instance.

Exceptions	
IInvalidRequest	The help window is not visible.

sizeTo Changes the size of the help cover page window, within which all other IPF windows are displayed. Call this function only when the help window is visible.

```
virtual IHelpWindow&
  sizeTo( const ISize& size );
```

Win	PM	Motif
Y	Y	Y

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect. You can use IHelpWindow::moveSizeTo (p. 454) to change both the size and position of the single native Windows help instance.

Exceptions	
IInvalidRequest	The help window is not visible.

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

IHelpWindow defines objects of the nested class IHelpWindow::Style (p. 464). You can use these styles with IHelpWindow::setDefaultStyle (p. 451).

classDefaultStyle

Specifies the original default style for this class, which is IHelpWindow::ipfCompatible (p. 456), except on Windows where it is set to IHelpWindow::noStyle (p. 457), meaning native Windows help.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

ipfCompatible

Specifies that IHelpWindow should use IPF to display application help information. Except for the Windows environment, this style is always on. For Windows, the default is IHelpWindow::noStyle (p. 457), therefore native Windows help is used. To use IPF on Windows, call IHelpWindow::setDefaultStyle (p. 451) with this style before constructing your IHelpWindow object or specify this style on the IHelpWindow constructor.

```
static const Style
  ipfCompatible;
```

Win	PM	Motif
Y	*Y*	*Y*

noStyle You use this style in the Windows environment to specify that native Windows help should be used to show help information. You only need to use this style if you have previously changed the default style to ipfCompatible (p. 456). You change your default IHelpWindow style by calling, IHelpWindow::setDefaultStyle (p. 451).

In all other environments, using this style has no effect.

```
static const Style
  noStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IWindow		
activeColorId	**disabledForegroundColorId**	**hiliteForegroundColorId**
backgroundColorId	**enableId**	**inactiveColorId**
borderColorId	**focusId**	**positionId**
commandId	**fontId**	**shadowColorId**
deleteId	**foregroundColorId**	**sizeId**
disabledBackgroundColorId	**hiliteBackgroundColorId**	**systemCommandId**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IHelpWindow contains the following nested classes:

IHelpWindow::Settings (see page 459)
IHelpWindow::Style (see page 464)

HelpType

```
HelpType {
    index,          general,          contents,         keys,
    usingHelp,      using = usingHelp
    };
```

Use these enumerators to identify a type of help window or help panel:

© IBM Corp. 1992, 1996

index The help Index window.

general The general help panel for the current frame window.

contents The help file Table of Contents window.

keys The keys help panel.

usingHelp The Using Help panel.

Win If you are using native Windows help, HelpType values have the following meanings:

index The Windows help Finder window that displays an index and provides searching capability.

general The general help panel for the current frame window.

contents The help file Table of Contents window.

keys The keys help panel.

usingHelp The Windows help Help-on-Help window.

IHelpWindow::Settings

Derivation
IBase
IHelpWindow::Settings

Inherited By None.

Header File ihelp.hpp

Members

Member	Page	Member	Page
Constructor	459	setMenuBar	462
setAccelerator	461	Settings	459
setAccelResLibrary	462	setTitle	462
setHelpResLibrary	460	setTutorial	461
setHelpTable	460	setUsingHelp	461
setLibraries	460		

The IHelpWindow::Settings class provides initial information for the constructor of the class IHelpWindow (p. 442). You can create a Settings object and call some or all of its member functions to set data before passing the object to the IHelpWindow constructor that accepts a Settings object.

IHelpWindow has individual functions to set all of the information that can be provided through a Settings object, with the exception of a tutorial name, the menu bar for the help window, and accelerator keys for the help window. If you need to customize your help support with these features, you must use this Settings class.

Win The tutorial name, the menu bar for the help window, and accelerator keys for the help window are not supported for native Windows help. Setting these values will have no effect for that style of IHelpWindow object.

Public Functions

Constructors

You can construct and destruct objects of this class.

Settings Create a settings object using the default constructor, which accepts no parameters.

```
Settings();
```

Win	PM	Motif
Y	*Y*	*Y*

© Copyright IBM Corp. 1992, 1996

Help Information

This class allows you to identify much of the help information that the user sees, which includes help panels and a tutorial program.

setHelpResLibrary

Sets the resource dynamic link library (DLL) that contains the help table resource. The default location of the resource library is the executable file.

Use this function in conjunction with the function setHelpTable (p. 460).

```
Settings&
  setHelpResLibrary( const char* helpTableResLibrary );
```

Win	PM	Motif
Y	Y	I

PM If you want to search the LIBPATH for the DLL, specify *helpTableResLibrary* without a path or extension.

Motif This function has no effect in AIX.

setHelpTable

Sets the help table to use for satisfying user requests for contextual and general help. Entries in the help table associate a help panel with a given application frame window or control.

```
Settings&
  setHelpTable( unsigned long helpTable );
```

Win	PM	Motif
Y	Y	Y

helpTable Identifier of a help table resource in a resource library. Use the function setHelpResLibrary (p. 460) to identify a resource dynamic link library (DLL).

setLibraries

Sets a library or list of libraries for IPF to reference when looking for help panels to display. If you specify more than one library name, separate them with a space character. This function deals with binary help files that have an .hlp extension.

```
Settings&
  setLibraries( const char* helpLibraryNames );
```

Win	PM	Motif
Y	Y	Y

PM If you want to search the HELP path for the help library files, specify the file names without a directory or path.

Win When using the native Windows help style for IHelpWindow, you should only pass a single file name on the setLibraries call. The file name should identify the help file that was compiled with the Windows help compiler from a Rich Text Format (RTF) source file.

setTutorial Sets the name of the application's tutorial. If you set a tutorial name, IPF provides a **Tutorial** choice on the menu bar of the help window. You process this menu choice by overriding the function IHelpHandler::showTutorial (p. 427).

```
Settings&
  setTutorial( const char* tutorial );
```

Win	PM	Motif
I	*Y*	*Y*

setUsingHelp Sets the help panel to use instead of the default Using Help panel provided by IPF. IPF displays this help panel when you call IHelpWindow::show (p. 449), passing the IHelpWindow::using (p. 457) enumerator.

```
Settings&
  setUsingHelp( unsigned long panelId );
```

Win	PM	Motif
Y	*Y*	*Y*

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function will change the help panel that is displayed when calling IHelpWindow::show (p. 449) with the HelpType of usingHelp.

Help Window Customization

You can customize the help information displayed by your application by changing aspects of the cover page window's appearance or behavior, including its menu bar, title bar text, and accelerator keys. The cover page window is the main help window within which all other IPF windows are displayed.

setAccelerator

Sets the accelerator keys that the help window will use instead of the default accelerator keys provided by IPF. If you provide a customized menu bar for the help window using the function setMenuBar (p. 462), you may need to provide your own accelerator table to match it.

```
Settings&
  setAccelerator( unsigned long acceleratorId );
```

Win	PM	Motif
Y	*Y*	*I*

acceleratorId
Identifier of an accelerator table resource in a resource library. Use the function setAccelResLibrary (p. 462) to identify a resource dynamic link library (DLL).

PM OS/2 requires that if the accelerator table is customized, then you must also customize the menu bar via IHelpWindow::Settings::setMenuBar.

© IBM Corp. 1992, 1996

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Motif This function has no effect in AIX.

setAccelResLibrary

Sets the resource dynamic link library (DLL) that contains the menu bar or accelerator table resources or both for the help window. The default location of the resource library is the executable file.

Use this function in conjunction with the functions setMenuBar (p. 462) and setAccelerator (p. 461).

```
Settings&
  setAccelResLibrary( const char* menuAccelResLibrary );
```

Win	PM	Motif
Y	Y	I

PM If you want to search the LIBPATH for the DLL, specify *menuAccelResLibrary* without a path or extension.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Motif This function has no effect in AIX.

setMenuBar

Sets the menu bar that the help window will use instead of the default menu bar provided by IPF. You may want to provide a customized menu bar to provide additional menu choices or for national language purposes.

```
Settings&
  setMenuBar( unsigned long menuBarId );
```

Win	PM	Motif
Y	Y	Y

menuBarId Identifier of a menu bar resource in a resource library. Use the function setAccelResLibrary (p. 462) to identify a resource dynamic link library (DLL).

PM OS/2 requires that if the menu bar is customized, then you must also customize the accelerator table via IHelpWindow::Settings::setAccelerator().

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

setTitle

Sets the title bar text for the help window.

1 `Settings&`
`  setTitle( const IResourceId& titleId );`

Win	PM	Motif
Y	Y	Y

You specify the text as a resource ID in a string table.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

2 `Settings&`
`  setTitle( const char* titleText );`

Win	PM	Motif
Y	Y	Y

You specify the text as a character string.

Win If you are using native Windows help (that is, you did not construct your IHelpWindow object with the IHelpWindow::ipfCompatible (p. 456) style), this function has no effect.

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IHelpWindow::Style

Derivation

IBase
IBitFlag
IHelpWindow::Style

Inherited By None.

Header File ihelp.hpp

The nested class IHelpWindow::Style defines the valid styles used for the IHelpWindow (p. 442) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IHighEventParameter

Derivation IBase
IHighEventParameter

Inherited By None.

Header File ievtdat2.hpp

The IHighEventParameter class encapsulates the message parameter (MPARAM) and result (MRESULT) data of an event.

IHighEventParameter is an alias of the IEventData class. See IEventData (p. 313) for more information.

In addition, IHighEventParameter is interchangeable with the following classes:

IEventParameter1 (p. 319)
IEventParameter2 (p. 320)
IEventResult (p. 321)
ILowEventParameter (p. 524)

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IKey

Derivation

IBase
IKey

Inherited By None.

Header File ikey.hpp

Members

Member	Page	Member	Page
alt	467	kF22	471
ctrl	467	kF23	471
kAlt	467	kF24	471
kAltGraf	468	kF3	471
kBackSpace	468	kF4	471
kBackTab	468	kF5	471
kBreak	468	kF6	471
kCapsLock	468	kF7	471
kCtrl	468	kF8	471
kDelete	468	kF9	472
kDown	468	kHome	472
kEnd	469	kInsert	472
kEnter	469	kLeft	472
kEsc	469	kNewLine	472
kF1	469	kNoKey	472
kF10	469	kNumLock	472
kF11	469	kPageDown	472
kF12	469	kPageUp	473
kF13	469	kPause	473
kF14	469	kRight	473
kF15	470	kScrollLock	473
kF16	470	kShift	473
kF17	470	kSpace	473
kF18	470	kSysRq	473
kF19	470	kTab	473
kF2	470	kUp	473
kF20	470	noModifier	467
kF21	470	shift	467

The IKey class contains common keyboard-related elements that can be used in more than one class.

You cannot create objects of this class, because IKey has no public constructors.

© Copyright IBM Corp. 1992, 1996

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Public Data

Key Modifiers

Use the KeyModifier (p. 475) objects to specify a combination of auxiliary keys. You can use these objects with the IAcceleratorKey (p. 22) class.

alt An IKey::KeyModifier (p. 475) object that represents the Alt key.

```
static const KeyModifier
  alt;
```

Win	PM	Motif
Y	*Y*	*N*

ctrl An IKey::KeyModifier (p. 475) object that represents the Ctrl key.

```
static const KeyModifier
  ctrl;
```

Win	PM	Motif
Y	*Y*	*N*

noModifier An IKey::KeyModifier (p. 475) object that represents the absence of the Alt, Ctrl, and Shift keys.

```
static const KeyModifier
  noModifier;
```

Win	PM	Motif
Y	*Y*	*N*

shift An IKey::KeyModifier (p. 475) object that represents the Shift key.

```
static const KeyModifier
  shift;
```

Win	PM	Motif
Y	*Y*	*N*

Virtual Keys

IKey provides a set of VirtualKey (p. 474) values that you can use as virtual key codes in the IAcceleratorKey class.

kAlt Represents the Alt key.

© IBM Corp. 1992, 1996

IKey

```
static const VirtualKey
  kAlt;
```

Win	PM	Motif
Y	*Y*	*N*

kAltGraf Represents the AltGraf key.

Note: This key is not available on all keyboards.

```
static const VirtualKey
  kAltGraf;
```

Win	PM	Motif
N	*Y*	*N*

kBackSpace Represents the Backspace key.

```
static const VirtualKey
  kBackSpace;
```

Win	PM	Motif
Y	*Y*	*N*

kBackTab Represents the back tab key.

Note: This is equivalent to the Tab key with the Shift modifier.

```
static const VirtualKey
  kBackTab;
```

Win	PM	Motif
N	*Y*	*N*

kBreak Represents the Break key.

```
static const VirtualKey
  kBreak;
```

Win	PM	Motif
Y	*Y*	*N*

kCapsLock Represents the Caps Lock key.

```
static const VirtualKey
  kCapsLock;
```

Win	PM	Motif
Y	*Y*	*N*

kCtrl Represents the Ctrl key.

```
static const VirtualKey
  kCtrl;
```

Win	PM	Motif
Y	*Y*	*N*

kDelete Represents the Delete key.

```
static const VirtualKey
  kDelete;
```

Win	PM	Motif
Y	*Y*	*N*

kDown Represents the down arrow key.

```
static const VirtualKey
  kDown;
```

Win	PM	Motif
Y	*Y*	*N*

kEnd Represents the End key.

```
static const VirtualKey
  kEnd;
```

Win	PM	Motif
Y	*Y*	*N*

kEnter Represents the Enter key.

```
static const VirtualKey
  kEnter;
```

Win	PM	Motif
Y	*Y*	*N*

kEsc Represents the Esc key.

```
static const VirtualKey
  kEsc;
```

Win	PM	Motif
Y	*Y*	*N*

kF1 Represents the F1 key.

```
static const VirtualKey
  kF1;
```

Win	PM	Motif
Y	*Y*	*N*

kF10 Represents the F10 key.

```
static const VirtualKey
  kF10;
```

Win	PM	Motif
Y	*Y*	*N*

kF11 Represents the F11 key.

```
static const VirtualKey
  kF11;
```

Win	PM	Motif
Y	*Y*	*N*

kF12 Represents the F12 key.

```
static const VirtualKey
  kF12;
```

Win	PM	Motif
Y	*Y*	*N*

kF13 Represents the F13 key.

```
static const VirtualKey
  kF13;
```

Win	PM	Motif
Y	*Y*	*N*

kF14 Represents the F14 key.

© IBM Corp. 1992, 1996

```
static const VirtualKey
  kF14;
```

Win	PM	Motif
Y	*Y*	*N*

kF15 Represents the F15 key.

```
static const VirtualKey
  kF15;
```

Win	PM	Motif
Y	*Y*	*N*

kF16 Represents the F16 key.

```
static const VirtualKey
  kF16;
```

Win	PM	Motif
Y	*Y*	*N*

kF17 Represents the F17 key.

```
static const VirtualKey
  kF17;
```

Win	PM	Motif
Y	*Y*	*N*

kF18 Represents the F18 key.

```
static const VirtualKey
  kF18;
```

Win	PM	Motif
Y	*Y*	*N*

kF19 Represents the F19 key.

```
static const VirtualKey
  kF19;
```

Win	PM	Motif
Y	*Y*	*N*

kF2 Represents the F2 key.

```
static const VirtualKey
  kF2;
```

Win	PM	Motif
Y	*Y*	*N*

kF20 Represents the F20 key.

```
static const VirtualKey
  kF20;
```

Win	PM	Motif
Y	*Y*	*N*

kF21 Represents the F21 key.

```
static const VirtualKey
  kF21;
```

Win	PM	Motif
Y	*Y*	*N*

kF22 Represents the F22 key.

```
static const VirtualKey
  kF22;
```

Win	PM	Motif
Y	*Y*	*N*

kF23 Represents the F23 key.

```
static const VirtualKey
  kF23;
```

Win	PM	Motif
Y	*Y*	*N*

kF24 Represents the F24 key.

```
static const VirtualKey
  kF24;
```

Win	PM	Motif
Y	*Y*	*N*

kF3 Represents the F3 key.

```
static const VirtualKey
  kF3;
```

Win	PM	Motif
Y	*Y*	*N*

kF4 Represents the F4 key.

```
static const VirtualKey
  kF4;
```

Win	PM	Motif
Y	*Y*	*N*

kF5 Represents the F5 key.

```
static const VirtualKey
  kF5;
```

Win	PM	Motif
Y	*Y*	*N*

kF6 Represents the F6 key.

```
static const VirtualKey
  kF6;
```

Win	PM	Motif
Y	*Y*	*N*

kF7 Represents the F7 key.

```
static const VirtualKey
  kF7;
```

Win	PM	Motif
Y	*Y*	*N*

kF8 Represents the F8 key.

© IBM Corp. 1992, 1996

```
static const VirtualKey
  kF8;
```

Win	PM	Motif
Y	*Y*	*N*

kF9 Represents the F9 key.

```
static const VirtualKey
  kF9;
```

Win	PM	Motif
Y	*Y*	*N*

kHome Represents the Home key.

```
static const VirtualKey
  kHome;
```

Win	PM	Motif
Y	*Y*	*N*

kInsert Represents the Insert key.

```
static const VirtualKey
  kInsert;
```

Win	PM	Motif
Y	*Y*	*N*

kLeft Represents the left arrow key.

```
static const VirtualKey
  kLeft;
```

Win	PM	Motif
Y	*Y*	*N*

kNewLine Represents the new line key.

Note: This key often has the label *Enter* on its key top.

```
static const VirtualKey
  kNewLine;
```

Win	PM	Motif
Y	*Y*	*N*

kNoKey Represents the absence of a virtual key.

```
static const VirtualKey
  kNoKey;
```

Win	PM	Motif
Y	*Y*	*N*

kNumLock Represents the Num Lock key.

```
static const VirtualKey
  kNumLock;
```

Win	PM	Motif
Y	*Y*	*N*

kPageDown Represents the Page Down key.

```
static const VirtualKey
  kPageDown;
```

Win	PM	Motif
Y	*Y*	*N*

kPageUp Represents the Page Up key.

```
static const VirtualKey
  kPageUp;
```

Win	PM	Motif
Y	*Y*	*N*

kPause Represents the Pause key.

```
static const VirtualKey
  kPause;
```

Win	PM	Motif
Y	*Y*	*N*

kRight Represents the right arrow key.

```
static const VirtualKey
  kRight;
```

Win	PM	Motif
Y	*Y*	*N*

kScrollLock Represents the Scroll Lock key.

```
static const VirtualKey
  kScrollLock;
```

Win	PM	Motif
Y	*Y*	*N*

kShift Represents the Shift key.

```
static const VirtualKey
  kShift;
```

Win	PM	Motif
Y	*Y*	*N*

kSpace Represents the Space bar.

```
static const VirtualKey
  kSpace;
```

Win	PM	Motif
Y	*Y*	*N*

kSysRq Represents the SysRq key.

```
static const VirtualKey
  kSysRq;
```

Win	PM	Motif
N	*Y*	*N*

kTab Represents the Tab key.

```
static const VirtualKey
  kTab;
```

Win	PM	Motif
Y	*Y*	*N*

kUp Represents the up arrow key.

© IBM Corp. 1992, 1996

```
static const VirtualKey
  kUp;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IKey contains the following nested classes:

IKey::KeyModifier (see page 475)

Nested Type Definitions

VirtualKey

```
typedef unsigned long VirtualKey;
```

This typedef helps document that the VirtualKey values provided by this class can be used as virtual key codes in the IAcceleratorKey (p. 22) class. The use of this typedef also allows you to use virtual key codes not provided by IKey with IAcceleratorKey.

IKey::KeyModifier

Derivation

IBase
  IBitFlag
    IKey::KeyModifier

Inherited By None.

Header File ikey.hpp

The nested class IKey::KeyModifier represents a modifier or auxiliary key that when combined with a base key, defines a complete keystroke. The class IKey (p. 466) defines IKey::KeyModifier objects (p. 467).

Like window style objects, IKey::KeyModifier objects can be combined using the bitwise OR operator. You can test for the presence of a specific IAcceleratorKey::KeyModifier object using the bitwise AND operator.

You can use IKey::KeyModifier objects when constructing an IAcceleratorKey object (p. 25) or when defining an accelerator key with the IAcceleratorKey::setKey (p. 28) function. The function IAcceleratorKey::keyModifier (p. 27) returns an IKey::KeyModifier object.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

© Copyright IBM Corp. 1992, 1996

IKey::KeyModifier

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IKeyboardConnectionTo

Derivation

IBase
  IVBase
    IHandler
      IKeyboardHandler
        IKeyboardConnectionTo

Inherited By None.

Header File ikeyhdr.hpp

Members

Member	Page	Member	Page
Constructor	478	virtualKeyPress	479
characterKeyPress	479	~IKeyboardConnectionTo	478
setVirtualKeyMemberFn	478		

The IKeyboardConnectionTo class is a template class, derived from IKeyboardHandler (p. 490), that processes keyboard events. This class allows you to process characterKeyPress and virtualKeyPress keyboard events in objects that do not derive from IKeyboardHandler without having to manually derive from IKeyboardHandler, override the characterKeyPress (p. 493) and virtualKeyPress (p. 493) functions, and pass the events to the object.

To use the IKeyboardConnectionTo class to process characterKeyPress events, follow these steps:

1. Instantiate the IKeyboardConnectionTo template with a class containing a member function whose signature and behavior are the same as IKeyboardHandler::characterKeyPress (p. 493).
2. Construct an object of the new template class by passing the object and the address of the member function handling the keyboard events.
3. Attach the template keyboard handler by using IHandler::handleEventsFor (p. 413) to pass the appropriate window to the keyboard handler.

You can additionally process virtualKeyPress events by calling the setVirtualKeyMember function with the address of a function that contains the same signature and behavior as IKeyboardHandler::virtualKeyPress (p. 493). To process only virtualKeyPress events, pass a null value for the address of the member function on the IKeyboardConnectionTo constructor.

© Copyright IBM Corp. 1992, 1996

IKeyboardConnectionTo

You use an IKeyboardConnectionTo object anywhere you use an IKeyboardHandler object. See the class description of IKeyboardHandler (p. 490) for a description of the uses and limitations of these classes.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign objects of this class.

IKeyboardConnectionTo

Constructs the IKeyboardConnectionTo object.

```
IKeyboardConnectionTo( ATarget& target,
                       KeyboardMemberFn characterKeyMemberFn );
```

Win	PM	Motif
Y	N	N

The parameters are the following:

target	The object to receive keyboard events.
characterKeyMemberFn	The address of the member function to receive character key press events.

~IKeyboardConnectionTo

```
virtual
  ~IKeyboardConnectionTo();
```

Win	PM	Motif
Y	N	N

Function Registration

Use these members to register functions on the target object to receive keyboard events.

setVirtualKeyMemberFn

Sets the function to process virtual key press events.

```
IKeyboardConnectionTo < ATarget >&
  setVirtualKeyMemberFn( KeyboardMemberFn virtualKeyMemberFn );
```

Win	PM	Motif
Y	N	N

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Processing

These members are overridden to route virtual and character key press events to the target object provided on construction.

characterKeyPress

Overridden to route character key press events to the *characterKeyMemberFn* function of the target object if you provided one on construction.

```
virtual Boolean
  characterKeyPress( IKeyboardEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

virtualKeyPress

Overridden to route virtual key press events to the *virtualKeyMemberFn* function of the target object if you provided one by calling setVirtualKeyMemberFn.

```
virtual Boolean
  virtualKeyPress( IKeyboardEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

© IBM Corp. 1992, 1996

IKeyboardConnectionTo

Inherited Protected Functions

IKeyboardHandler		
characterKeyPress	dispatchHandlerEvent	key

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

KeyboardMemberFn

```
typedef Boolean (ATarget::*KeyboardMemberFn)(IKeyboardEvent&);
```

A pointer to a member function of Class ATarget that takes a reference to an IKeyboardEvent and returns a Boolean.

IKeyboardEvent

Derivation

IBase
  IVBase
    IEvent
      IKeyboardEvent

Inherited By None.

Header File ikeyevt.hpp

Members

Member	Page	Member	Page
Constructor	483	ulAltFlag	486
character	482	ulAltMask	486
isAltDown	484	ulCharacterFlag	487
isCharacter	484	ulCompositeFlag	487
isComposite	484	ulCtrlFlag	487
isCtrlDown	484	ulCtrlMask	486
isForComposite	484	ulForCompositeFlag	487
isInvalidComposite	484	ulInvalidCompositeFlag	487
isRepeat	484	ulRepeatFlag	487
isScanCode	485	ulScanCodeFlag	487
isShiftDown	485	ulShiftFlag	487
isUncombined	485	ulShiftMask	486
isUpTransition	485	ulUncombinedFlag	488
isVirtual	485	ulUpTransitionFlag	488
mixedCharacter	482	ulVirtualFlag	488
repeatCount	482	virtualKey	483
scanCode	483	~IKeyboardEvent	483

The IKeyboardEvent class represents keyboard-related events. A keyboard handler creates an IKeyboardEvent object when a user presses or releases a key.

You must verify that the character code value, scan code value, or virtual key value is valid before you use the value. Use isCharacter (p. 484), isScanCode (p. 485), and isVirtual (p. 485), respectively, to do so. This class provides similar functions to check if the Shift key, Ctrl key, or Alt key is pressed.

If, however, you are within the callback function IKeyboardHandler::characterKeyPress (p. 493), you do not need to check the character code value with isCharacter. The same is true for the scan code value from IKeyboardHandler::scanCodeKeyPress (p. 493) and the virtual key value from IKeyboardHandler::virtualKeyPress (p. 493).

© Copyright IBM Corp. 1992, 1996

Keyboard events are first dispatched to the window with the input focus, such as an entry field with the input cursor. If that window does not process the keyboard event, the event is dispatched to the owner window of the focus window. The event continues to be dispatched to the next owner window until a handler stops the processing or a window processes the keystroke itself. For example, an entry field processes character keys, but it does not process a Tab key, so the Tab key is passed on to the owner window of the entry field.

Public Functions

Accessors

These members allow you to query values from instances of this class.

character Returns the character data of the key if it is a single-byte character. Before calling this function, you must verify that the event contains character data by calling IKeyboardEvent::isCharacter (p. 484). If you expect double-byte data, use IKeyboardEvent::mixedCharacter (p. 482).

```
char
  character() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Exceptions	
IInvalidRequest	The keyboard event does not contain character data, or it contains a double-byte character.

mixedCharacter

Returns the character data of the key, whether it is a single- or double-byte character. Before calling this function, you must verify that the event contains character data by calling IKeyboardEvent::isCharacter (p. 484).

```
IString
  mixedCharacter() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif When working with DBCS characters, consider using the class IEditVerifyHandler instead of IKeyboardHandler (p. 490).

Exceptions	
IInvalidRequest	The keyboard event does not contain character data.

repeatCount Returns the number of repeated keys combined into this event.

```
unsigned long
  repeatCount() const;
```

Win	PM	Motif
Y	Y	Y

scanCode Returns the hardware scan code of the key. Before calling this function, you must verify that the event contains a scan code by calling IKeyboardEvent::isScanCode (p. 485).

```
unsigned long
  scanCode() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The keyboard event does not contain a scan code, or it represents a double-byte character.

virtualKey Returns the virtual key code of the key. Before calling this function, you must verify that the event contains a virtual key code by calling IKeyboardEvent::isVirtual (p. 485). The enumeration VirtualKey (p. 488) provides the return values.

```
VirtualKey
  virtualKey() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The keyboard event does not contain virtual key information.

Constructors

You can construct and destruct objects of this class.

IKeyboardEvent

Constructs an IKeyboardEvent object from the specified event. IKeyboardHandler::dispatchHandlerEvent (p. 492) constructs objects of this class from an object of the class IEvent (p. 304).

```
IKeyboardEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IKeyboardEvent

```
virtual
  ~IKeyboardEvent();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Testing

Call these testing members to determine the type of data stored in the event.

isAltDown Returns whether the **Alt** key is being held down.

```
Boolean
  isAltDown() const;
```

Win	PM	Motif
Y	Y	Y

isCharacter Returns whether the event represents use of a data key.

```
Boolean
  isCharacter() const;
```

Win	PM	Motif
Y	Y	Y

isComposite Returns whether the character is formed by combining this key with the previous nonescaping key.

```
Boolean
  isComposite() const;
```

Win	PM	Motif
N	Y	Y

isCtrlDown Returns whether the **Ctrl** key is being held down.

```
Boolean
  isCtrlDown() const;
```

Win	PM	Motif
Y	Y	Y

isForComposite

Returns whether the character is a nonescaping key, such as an accent, which the user intends to combine with the next character.

```
Boolean
  isForComposite() const;
```

Win	PM	Motif
Y	Y	Y

isInvalidComposite

Returns whether the character is not valid to be combined with the previous nonescaping key.

```
Boolean
  isInvalidComposite() const;
```

Win	PM	Motif
N	Y	Y

isRepeat Returns whether a key has been previously pressed and is being held down.

```
Boolean
  isRepeat() const;
```

Win	PM	Motif
Y	Y	Y

isScanCode Returns whether a hardware scan code is available.

```
Boolean
  isScanCode() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isShiftDown Returns whether the **Shift** key is being held down.

```
Boolean
  isShiftDown() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isUncombined

Returns whether this key is being pressed without any other keys being pressed.

```
Boolean
  isUncombined() const;
```

Win	PM	Motif
N	*Y*	*Y*

isUpTransition

Returns whether a previously pressed key is being released.

```
Boolean
  isUpTransition() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isVirtual Returns whether the event represents use of a virtual key.

```
Boolean
  isVirtual() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

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

© IBM Corp. 1992, 1996

IKeyboardEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Public Data

Implementation

These static flags test corresponding flags in the key event.

ulAltMask Used to implement the isAltDown (p. 484) check.

```
static unsigned long
  ulAltMask;
```

Win	PM	Motif
N	*N*	*Y*

ulCtrlMask Used to implement the isCtrlDown (p. 484) check.

```
static unsigned long
  ulCtrlMask;
```

Win	PM	Motif
N	*N*	*Y*

ulShiftMask Used to implement the isShiftDown (p. 485) check.

```
static unsigned long
  ulShiftMask;
```

Win	PM	Motif
N	*N*	*Y*

Protected Data

Implementation

These static flags test corresponding flags in the key event.

ulAltFlag Used to implement the isAltDown (p. 484) check.

```
static const unsigned long
  ulAltFlag;
```

Win	PM	Motif
Y	*Y*	*Y*

ulCharacterFlag

Used to implement the isCharacter (p. 484) check.

```
static const unsigned long
  ulCharacterFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulCompositeFlag

Used to implement the isComposite (p. 484) check.

```
static const unsigned long
  ulCompositeFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulCtrlFlag

Used to implement the isCtrlDown (p. 484) check.

```
static const unsigned long
  ulCtrlFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulForCompositeFlag

Used to implement the isForComposite (p. 484) check.

```
static const unsigned long
  ulForCompositeFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulInvalidCompositeFlag

Used to implement the isInvalidComposite (p. 484) check.

```
static const unsigned long
  ulInvalidCompositeFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulRepeatFlag

Used to implement the isRepeat (p. 484) check.

```
static const unsigned long
  ulRepeatFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulScanCodeFlag

Used to implement the isScanCode (p. 485) check.

```
static const unsigned long
  ulScanCodeFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulShiftFlag

Used to implement the isShiftDown (p. 485) check.

© IBM Corp. 1992, 1996

```
static const unsigned long
  ulShiftFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulUncombinedFlag

Used to implement the isUncombined (p. 485) check.

```
static const unsigned long
  ulUncombinedFlag;
```

Win	PM	Motif
N	*Y*	*Y*

ulUpTransitionFlag

Used to implement the isUpTransition (p. 485) check.

```
static const unsigned long
  ulUpTransitionFlag;
```

Win	PM	Motif
Y	*Y*	*Y*

ulVirtualFlag

Used to implement the isVirtual (p. 485) check.

```
static const unsigned long
  ulVirtualFlag;
```

Win	PM	Motif
N	*Y*	*Y*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

VirtualKey

```
VirtualKey {
    esc,            tab,            backTab,        space,
    backSpace,      enter,          newLine,        shift,
    ctrl,           altGraf,        insert,         deleteKey,
    home,           end,            pageUp,         pageDown,
    left,           right,          up,             down,
    capsLock,       numLock,        scrollLock,     pause,
    sysRq,          breakKey,       f2,             f3,
    f4,             f5,             f6,             f7,
    f8,             f9,             f11,            f12,
    f13,            f14,            f15,            f16,
    f17,            f18,            f19,            f20,
    f21,            f22,            f23,            f24,
    button1,        button2,        button3,        other,
    endDrag,        firstDBCS,      lastDBCS,       firstUser,
    lastUser,       f1,             f10,            f25,
    f26,            f27,            f28,            f29,
```

```
f30,              f31,              f32,              f33,
f34,              f35,              pf1,              pf2,
pf3,              pf4,              clear,            select,
print,            execute,          undo,             redo,
menu,             find,             cancel,           help,
modeSwitch,       alt,              shiftLock,        begin,
meta,             super,            hyper,            multiKey,
kanji,            muhenkan,         henkanMode,       romaji,
hiragana,         katakana,         hiraganaKatakana, zenkaku,
hankaku,          zenkakuHankaku,   touroku,          massyo,
kanaLock,         kanaShift,        eisuShift,        eisuToggle
};
```

The preceding enumerators list the possible virtual key values that IKeyboardEvent::virtualKey (p. 483) can return. The User Interface Class Library provides enumerators for a generic keyboard, not just an IBM keyboard.

PM The virtual key codes *f1*, *f10*, and *alt* do not appear in this list. Presentation Manager intercepts F1 and F10 key presses, and Alt releases, and processes them as system accelerators. Because of this, you never see them as keyboard events.

The following enumerators also are not supported: *f25*, *f26*, *f27*, *f28*, *f29*, *f30*, *f31*, *f32*, *f33*, *f34*, *f35*, *pf1*, *pf2*, *pf3*, *pf4*, *clear*, *select*, *print*, *execute*, *undo*, *redo*, *menu*, *find*, *cancel*, *help*, *modeSwitch*, *alt*, *shiftLock*, *begin*, *meta*, *super*, *hyper*, *multiKey*, *kanji*, *muhenkan*, *henkanMode*, *romaji*, *hiragana*, *katakana*, *hiraganaKatakana*, *zenkaku*, *hankaku*, *zenkakuHankaku*, *touroku*, *massyo*, *kanaLock*, *kanaShift*, *eisuShift*, and *eisuToggle*.

© IBM Corp. 1992, 1996

IKeyboardHandler

Derivation

IBase
IVBase
IHandler
IKeyboardHandler

Inherited By IKeyboardConnectionTo

Header File ikeyhdr.hpp

Members

Member	Page	Member	Page
Constructor	491	scanCodeKeyPress	493
characterKeyPress	493	virtualKeyPress	493
dispatchHandlerEvent	492	~IKeyboardHandler	492
key	493		

The IKeyboardHandler class processes all types of keyboard events.

Create a handler derived from IKeyboardHandler and attach it to any window whose keyboard events you want to handle. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the window to the keyboard handler.

You can attach IKeyboardHandlers to the following controls:

- ICheckBox (p. 146)
- IComboBox (p. 182)
- IContainerControl (Vol. III)
- IEntryField (p. 271)
- IListBox (p. 495)
- IMultiLineEdit (p. 641)
- INotebook (Vol. III)
- IProgressIndicator (p. 743)
- IRadioButton (p. 786)
- ISlider (p. 874)
- INumericSpinButton (p. 672)
- ITextSpinButton (p. 963)

When the keyboard handler receives a keyboard event, it creates an object of IKeyboardEvent (p. 481) and routes that object to the appropriate IKeyboardHandler

© Copyright IBM Corp. 1992, 1996

virtual function. You can override these virtual functions to supply your own specialized processing of these events.

A keyboard event continues to travel up the owner chain until either a handler stops it or the key is processed by the window itself. As a result, an IKeyboardHandler object should not stop the processing of any key event that it does not specifically process because the key has the potential of being handled by any window in the owner chain, such as the frame window.

The return value from the virtual functions specifies whether the keyboard event is passed on for additional processing, as follows:

true The keyboard event requires no additional processing. Do not pass it to another handler.

false Pass the keyboard event to the next handler for additional processing, as follows:

- If there is another handler for the focus window, pass the keyboard event to the next handler.
- If this is the last handler for the focus window, call IWindow::defaultProcedure (p. 1082) to process the keyboard event. This call might cause the keyboard event to be dispatched to the focus window's owner window.

PM You can attach IKeyboardHandlers to I3StateCheckBox (p. 8) objects.

Motif Some types of key processing, such as peeking at or modifying key input prior to its being stored in a control, are best handled by using the class IEditVerifyHandler.

Public Functions

Constructors

You can construct and destruct objects of this class.

IKeyboardHandler

Provides the default constructor.

```
IKeyboardHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

IKeyboardHandler

~IKeyboardHandler

```
virtual
  ~IKeyboardHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IKeyboardHandler object processes only keyboard-related events.

dispatchHandlerEvent

If a keyboard event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

A keyboard handler contains event-processing members that you can use to provide application-specific processing for a keyboard event. Override at least one of these virtual functions in a derived class.

Note: Some keyboard events are routed to more than one of these virtual functions.

characterKeyPress

Implemented by derived classes to handle character key presses. When using this callback function, you do not have to verify the character code value using IKeyboardEvent::isCharacter (p. 484). This function can only be called for key-down transitions or if the user holds down a key.

```
virtual Boolean
  characterKeyPress( IKeyboardEvent& event );
```

Win	PM	Motif
Y	Y	Y

key

Implemented by derived classes to handle any and all unprocessed key events. This function lets you process every keystroke as it is entered as well as key-down and key-up transitions.

```
virtual Boolean
  key( IKeyboardEvent& event );
```

Win	PM	Motif
Y	Y	Y

scanCodeKeyPress

Implemented by derived classes to handle scan code key presses. When using this callback function, you do not have to verify that the event contains a scan code value using IKeyboardEvent::isScanCode (p. 485). This function can only be called for key-down transitions or if the user holds down a key.

```
virtual Boolean
  scanCodeKeyPress( IKeyboardEvent& event );
```

Win	PM	Motif
Y	Y	Y

virtualKeyPress

Implemented by derived classes to handle virtual key presses. When using this callback function, you do not have to verify that the event contains a virtual key value using IKeyboardEvent::isVirtual (p. 485). This function can only be called for key-down transitions or if the user holds down a key.

```
virtual Boolean
  virtualKeyPress( IKeyboardEvent& event );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

© IBM Corp. 1992, 1996

IKeyboardHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IListBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IBaseListBox
            IListBox

Inherited By None.

Header File ilistbox.hpp

Members

Member	Page	Member	Page
Constructor	501	defaultStyle	503
add	496	remove	502
addAscending	497	removeAll	502
addAsFirst	498	removeAt	502
addAsLast	499	removeId	505
addAsNext	499	replaceAt	502
addDescending	500	setDefaultStyle	503
addId	505	~IListBox	501
classDefaultStyle	505		

The IListBox class extends the list box function in IBaseListBox to include adding, removing, and replacing list box items. List boxes always have vertical scroll bars. Optionally, they can have horizontal scroll bars.

You can enable a list box for the following types of item-selection techniques:

single selection
The user can select only one item at a time.

multiple selection
The user can select any number of items or not select any.

extended selection
The user can extend selection to more than one item.

These three techniques are mutually exclusive in one list box.

A list box operates as if it is a 0-based array of items. The item index requested or returned is the 0-based index number of the location of the item in question.

Handlers derived from the following classes handle events for IListBox:

© Copyright IBM Corp. 1992, 1996

- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IListBoxDrawItemHandler (p. 513)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISelectHandler (p. 851)

Public Functions

Add Items

Use these members to add items to the list box. You can add items to the list box at the following positions:

- A specified location given by an index or cursor
- The first item
- The last item
- The next item following a cursor
- In descending sort order
- In ascending sort order

Represent the new text item by either a resource identifier or the text string itself.

add Inserts the line of text at a specified location in the list box.

1
```
virtual IListBox&
  add( const char* item,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box at the cursor position and sets the cursor to the inserted item.

item The text you want to insert.
cursor A list box cursor object. The cursor's position identifies where to insert the item in the list.

2
```
virtual unsigned long
  add( unsigned long index,
       const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts text into the list box and returns the index of the item.

index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

text The text you want to insert.

3
```
virtual unsigned long
  add( unsigned long index,
       const char * const* itemList,
       unsigned long count = 1 );
```

Win	PM	Motif
Y	*Y*	*Y*

Inserts items corresponding to the itemList text strings into the list box and returns the index of the first item inserted.

index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

itemList An array of character strings.

count Number of text items in itemList. The default is 1 item.

4
```
virtual unsigned long
  add( unsigned long index,
       const IResourceId& item );
```

Win	PM	Motif
Y	*Y*	*Y*

Inserts the text corresponding to the resource item into the list box and returns the index of the item.

index Index position of the item in the list. If the index is greater than the number of items, the text is inserted at the end of the list.

item Resource ID of the text you want to insert in the list.

5
```
virtual IListBox&
  add( const IResourceId& text,
       Cursor& cursor );
```

Win	PM	Motif
Y	*Y*	*Y*

Inserts text into the list box at the cursor position and sets the cursor to the inserted item.

text Resource ID of the text you want to insert into the list.
cursor A list box cursor object. The cursor's position identifies where to insert the item in the list.

addAscending

Inserts the line of text in ascending sort order.

1
```
virtual unsigned long
  addAscending( const char* text );
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

Inserts text into the list box in ascending sort order and returns the index of the inserted item.

text The text you want to insert.

2
```
virtual unsigned long
  addAscending( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, into the list box in ascending sort order and returns the index of the inserted item.

item Resource ID of the text you want to insert in the list.

addAsFirst

Inserts the line of text as the first item in the list box.

1
```
virtual unsigned long
  addAsFirst( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the first item in the list box.

text The text you want to insert.

2
```
virtual unsigned long
  addAsFirst( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the first item in the list box.

item Resource ID of the text you want to insert in the list.

3
```
virtual IListBox&
  addAsFirst( const char* item,
              Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the first item in the list box and puts the cursor on the inserted item.

item The text you want to insert.
cursor A list box cursor object. The cursor is set to the first item in the list.

4
```
virtual IListBox&
  addAsFirst( const IResourceId& text,
              Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box and puts the cursor on that item.

text Resource ID of the text you want to insert in the list.
cursor A list box cursor object. The cursor is set to the first item in the list.

addAsLast Inserts the line of text as the last item in the list box.

1
```
virtual unsigned long
  addAsLast( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box.

item Resource ID of the text you want to insert in the list.

2
```
virtual unsigned long
  addAsLast( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the last item in the list box.

text The text you want to insert.

3
```
virtual IListBox&
  addAsLast( const char* item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text as the last item in the list box and puts the cursor on that item.

item The text you want to insert.
cursor A list box cursor object. The cursor is set to the last item in the list.

4
```
virtual IListBox&
  addAsLast( const IResourceId& text,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text, corresponding to the resource identifier, as the last item in the list box and puts the cursor on that item.

text Resource ID of the text you want to insert in the list.
cursor A list box cursor object. The cursor is set to the last item in the list.

addAsNext Inserts the line of text in the list box after the cursor.

© IBM Corp. 1992, 1996

1
```
virtual IListBox&
  addAsNext( const char* item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts the text after the current cursor position in the list box and places the cursor on that item.

item The text you want to insert.
cursor A list box cursor object. The cursor's position identifies where to insert the item in the list.

2
```
virtual IListBox&
  addAsNext( const IResourceId& text,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Inserts a new text item, corresponding to the resource identifier, after the current cursor position in the list box and places the cursor on that item.

text Resource ID of the text you want to insert in the list.
cursor A list box cursor object. The cursor's position identifies where to insert the item in the list.

addDescending

Inserts the line of text in descending sort order.

1
```
virtual unsigned long
  addDescending( const char* text );
```

Win	PM	Motif
Y	Y	Y

Inserts the line of text in descending sort order and returns the index of the item inserted.

text The text string to be inserted into the list box.

2
```
virtual unsigned long
  addDescending( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

Inserts the line of text, corresponding to the resource identifier, in descending sort order and returns the index of the item inserted.

item The resource ID of the text string.

Constructors

You can construct and destruct objects of this class.

IListBox

1 `IListBox( unsigned long id,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial = IRectangle ( ),`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the parent window, owner window, optional size and location, and optional style arguments.

id The ID of a list box control.

parent The parent window.

owner The owner window.

initial A rectangle for the list box control. It specifies the initial position and size of the IListBox you construct. The initial position is the lower-left corner of the list box relative to the lower-left corner of the parent window. The default is the rectangle constructed by the default IRectangle constructor. Optional.

style The initial style for the list box control. The classDefaultStyle is used if no style is provided. Optional.

2 `IListBox( unsigned long id,`
`IWindow* parent );`

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the parent window and a list box control ID.

id The ID of a list box control.
parent The parent window.

3 `IListBox( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	Y

You can create a list box control object using the handle of an existing list box control.

handle A window handle of an existing list box control.

~IListBox

© IBM Corp. 1992, 1996

```
virtual
  ~IListBox();
```

Win	PM	Motif
Y	Y	Y

Remove and Replace Items

Use these members to remove items from or replace items in the list box.

remove Removes the specified item from the list box and returns the count of items that remain.

index The index of the text string that is removed.

```
virtual unsigned long
  remove( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

removeAll Removes all items from the list box.

```
virtual IListBox&
  removeAll();
```

Win	PM	Motif
Y	Y	Y

removeAt Removes the item at the cursor and places the cursor at the next available valid item.

cursor A list box cursor.

```
virtual IListBox&
  removeAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

replaceAt Replaces the list box item at the cursor position. An invalid cursor causes an exception.

1

```
virtual IListBox&
  replaceAt( const char* item,
             const Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Replaces the item's string at the cursor with the new text.

item The text string.
cursor A list box cursor object.

2

```
virtual IListBox&
  replaceAt( const IResourceId& text,
             const Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Replaces the item's string at the cursor with the text corresponding to the resource identifier.

text The resource ID of the text string to replace in the list box.
cursor A list box cursor object.

Styles

Use these members to define, set, and retrieve the IListBox default style.

Note: The IListBox class inherits its style behavior from IBaseListBox except for the default style members described here.

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 505) unless you have changed the style using setDefaultStyle (p. 503).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent list boxes.

style Use the styles provided by IListBox Styles (p. 505) to specify the default list box style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IBaseListBox		
backgroundColor	enableNoAdjustPosition	minimumRows
convertToGUIStyle	enableNotification	numberOfSelections
count	isDrawItem	select
defaultStyle	isEmpty	selectAll
deselect	isExtendedSelect	selection
deselectAll	isHorizontalScroll	**setDefaultStyle**
disableDrawItem	isMultipleSelect	setItemHandle
disableExtendedSelect	isNoAdjustPosition	setItemHeight
disableMultipleSelect	isSelected	setItemText

© IBM Corp. 1992, 1996

IListBox

IBaseListBox		
disableNoAdjustPosition	itemHandle	setLayoutDistorted
elementAt	itemHeight	setMinimumCharacters
enableDrawItem	itemText	setMinimumRows
enableExtendedSelect	locateText	setTop
enableMultipleSelect	minimumCharacters	show

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 (p. 1044) are not shown.

Inherited Protected Functions

IBaseListBox		
calcMinimumSize	incrementChangeCount	registerCallbacks
changeCount	passEventToOwner	unregisterCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

These INotificationId members define the possible notifications that IListBox provides to its observers. The following events can be observed:

- Add a list box item.
- Remove a list box item.

addId This notification identifier is the notification IListBox objects provide their observers when an item is added to the list box. The number of items added is provided in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  addId;
```

Win	PM	Motif
Y	*Y*	*Y*

removeId This notification identifier is the notification IListBox objects provide their observers when an item is removed from the list box. The number of items removed is provided in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). If the value is 0, all items were removed.

```
static INotificationId const
  removeId;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

Use these members to define, set, and retrieve the IListBox default style.

Note: The IListBox class inherits its style behavior from IBaseListBox except for the default style members described here.

classDefaultStyle

Provides the original default style for this class, which is the following: IBaseListBox::horizontalScroll | IBaseListBox::noAdjustPosition | IWindow::visible.

If you do not specify either the extendedSelect or multipleSelect style, the list box is a single-selection list box by default.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

Inherited Public Data

IBaseListBox		
border3D	**enterId**	**horizontalScroll**
classDefaultStyle	**extendedSelect**	**multipleSelect**
drawItem	**first**	**noAdjustPosition**

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IListBox contains the following nested classes:

IListBox::Style (see page 508)

Nested Type Definitions

Cursor

```
typedef IBaseListBox::Cursor Cursor;
```

This typedef supports prior use of the IListBox::Cursor class, which now exists in IBaseListBox.

© IBM Corp. 1992, 1996

IListBox::Style

Derivation

IBase
IBitFlag
IListBox::Style

Inherited By None.

Header File ilistbox.hpp

The nested class IListBox::Style provides a set of valid styles for the IListBox (p. 495) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IListBoxDrawItemEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IDrawItemEvent
          IListBoxDrawItemEvent

Inherited By None.

Header File ilbdievt.hpp

Members

Member	Page	Member	Page
Constructor	509	setSelectionStateDrawn	510
isSelected	510	~IListBoxDrawItemEvent	510
isSelectionStateDrawn	510		

The IListBoxDrawItemEvent class provides event information for drawing items in a list box.

Public Functions

Constructors

You can construct and destruct objects of this class.

Although you can construct objects of this class, typically IListBoxDrawItemHandler::dispatchHandlerEvent (p. 516) creates objects of this class from an object of the class IEvent (p. 304).

IListBoxDrawItemEvent

Constructs an object of this class. The only way to construct an object of this class is from an object of the class IEvent (p. 304).

```
IListBoxDrawItemEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

IListBoxDrawItemEvent

~IListBoxDrawItemEvent

```
virtual
  ~IListBoxDrawItemEvent();
```

Win	PM	Motif
Y	Y	N

Event Information

Use these members to query information about the list box item.

isSelected If the list box item is selected, true is returned. Otherwise, false is returned.

```
Boolean
  isSelected() const;
```

Win	PM	Motif
Y	Y	N

Selection State

Use these members to manage the drawing of selection state emphasis.

isSelectionStateDrawn

Returns a flag indicating if the handler has drawn the list box item to indicate its selection status. This function returns the value set with setSelectionStateDrawn (p. 510). If the latter function has not been called, false is returned.

This value is ignored if drawItem (p. 516), selectItem (p. 516), or deselectItem (p. 516) of the IListBoxDrawItemHandler (p. 513) that is processing the event returns false. If the handler returns true and isSelectionStateDrawn returns true, the windowing system stops drawing the item. If the handler returns true and isSelectionStateDrawn returns false, the windowing system changes the highlighting of the item.

```
Boolean
  isSelectionStateDrawn() const;
```

Win	PM	Motif
Y	Y	N

setSelectionStateDrawn

Sets whether the handler has drawn the list box item to indicate its selection status. If the function has not been called, the handler has not drawn the item's selection status.

The value set by this function is ignored if drawItem (p. 516), selectItem (p. 516), or deselectItem (p. 516) of the IListBoxDrawItemHandler (p. 513) that is processing the event returns false. If the handler returns true and setSelectionStateDrawn is called with true, the windowing system stops drawing the item. If the handler returns true

and setSelectionStateDrawn is called with false, the windowing system changes the highlighting of the item.

```
IListBoxDrawItemEvent&
  setSelectionStateDrawn( Boolean drawn = true );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IDrawItemEvent		
itemId	itemPresSpaceHandle	itemRect

IControlEvent		
controlId		

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 Functions

IDrawItemEvent		
ownerItemData		

© IBM Corp. 1992, 1996

IListBoxDrawItemEvent

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IListBoxDrawItemHandler

Derivation

IBase
  IVBase
    IHandler
      IListBoxDrawItemHandler

Inherited By None.

Header File ilbdihdr.hpp

Members

Member	Page	Member	Page
Constructor	514	handleEventsFor	515
deselectItem	516	selectItem	516
dispatchHandlerEvent	516	setItemSize	517
drawItem	516	~IListBoxDrawItemHandler	514

The IListBoxDrawItemHandler class processes list box draw-item events.

Note: You must create the list box with the IListBox::drawItem style (see IListBox Styles (p. 505) for information about the drawItem style), or later call the function IBaseListBox::enableDrawItem (p. 92).

Create a handler derived from IListBoxDrawItemHandler and attach it to a list box or its owner window. You can do this by calling IListBoxDrawItemHandler::handleEventsFor (p. 515).

When the list box draw-item handler receives a list box draw-item event, it creates an IListBoxDrawItemEvent or IListBoxSizeItemEvent object and routes that object to the appropriate IListBoxDrawItemHandler virtual function. Override these virtual functions to supply your own specialized processing of a list box draw-item event. See IListBoxDrawItemEvent (p. 509) and IListBoxSizeItemEvent (p. 521) for information about these classes.

The return value from the virtual functions specifies whether the list box draw-item event should be passed on for additional processing, as follows:

© Copyright IBM Corp. 1992, 1996

Return Value	Meaning
true	The list box draw-item event requires no additional processing. Do not pass it to another handler. The windowing system will not draw the item, although it can highlight or unhighlight the item based on the value returned by IListBoxDrawItemEvent::setSelectionStateDrawn (p. 510).
false	The list box draw-item event requires additional processing. Pass the list box draw-item event to the next handler, as follows: • If there is another handler for the list box, pass the list box draw-item event to the next handler. • If this is the last handler for the list box, call the IWindow::dispatch function to dispatch the list box draw-item event to the list box's owner window. See dispatch (p. 1082) for information about that function. • If this is the last handler for the owner window, call the IWindow::defaultProcedure function to process the list box draw-item event. See defaultProcedure (p. 1082) for information about that function.

Public Functions

Constructors

You can construct and destruct objects of this class.

IListBoxDrawItemHandler

The only way to construct an object of this class is to use the default constructor, which does not accept any arguments.

```
IListBoxDrawItemHandler();
```

Win	PM	Motif
Y	*Y*	*N*

~IListBoxDrawItemHandler

```
virtual
  ~IListBoxDrawItemHandler();
```

Win	PM	Motif
Y	*Y*	*N*

Event Dispatching

Use these members in the dispatching of events by the handler. These members evaluate events for potential processing or are used to attach the handler to a list box or its owner window.

handleEventsFor

Attaches the handler to process item-drawing events for the specified window. You can attach the handler to a list box or to its owner window.

1
```
virtual IListBoxDrawItemHandler&
  handleEventsFor( IWindow* listBoxOwner );
```

Win	PM	Motif
Y	*Y*	*N*

2
```
virtual IListBoxDrawItemHandler&
  handleEventsFor( IListBox* listBox );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

Use these members in the dispatching of events by the handler. These members evaluate events for potential processing or are used to attach the handler to a list box or its owner window.

© IBM Corp. 1992, 1996

IListBoxDrawItemHandler

dispatchHandlerEvent

Calls the appropriate virtual function if a list box draw-item event is found.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

Event Processing

Use these members to process specific requests in support of drawing a list box item. You can override these virtual members in a derived class.

deselectItem Called when a list box item is deselected. Override this function if you want to remove the highlight to indicate a list item has been deselected. Do this only if the system default method is unacceptable. To prevent the system default method from altering the item, call IListBoxDrawItemEvent::setSelectionStateDrawn (p. 510), and return true.

```
virtual Boolean
  deselectItem( IListBoxDrawItemEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

drawItem Called when a list box item needs to be drawn or refreshed. Override this function to draw the list item. By default, the system applies selected-state emphasis (inverts the bits used to display the item) to indicate list-item selection. Unless the system default method is unacceptable, you do not have to draw the list item.

If you draw the list item yourself, return true from this function. You can additionally control whether the windowing system draws any highlighting to indicate whether the item is selected. To prevent the system default method from altering the highlighting of the item, call IListBoxDrawItemEvent::setSelectionStateDrawn (p. 510) before returning true.

The IListBoxDrawItemEvent class provides isSelected (p. 510) to query whether the list item is selected.

```
virtual Boolean
  drawItem( IListBoxDrawItemEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

selectItem Called when a list box item is selected. Override this function if you want to draw the highlighting that indicates the list item has been selected. Do this only if the system default of inverting the bits for a selected item is unacceptable. To prevent the system default method from altering the item, call IListBoxDrawItemEvent::setSelectionStateDrawn (p. 510) and return true.

```
virtual Boolean
  selectItem( IListBoxDrawItemEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

setItemSize Called if any of the following occurs:

- The list box is first created.
- This handler is attached to a list box.
- You call IBaseListBox::enableDrawItem (p. 92).
- Items are added or removed from the list box.

Override this function to set the size needed to draw a list box item. All items in a list box have the same height. The width you specify is only used if the list box has a horizontal scroll bar.

```
virtual Boolean
  setItemSize( IListBoxSizeItemEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IListBoxNotifyHandler

Derivation

IBase
IVBase
IHandler
IWindowNotifyHandler
IListBoxNotifyHandler

Inherited By None.

Header File ilistbnh.hpp

Members

Member	Page
Constructor	518
dispatchHandlerEvent	519
~IListBoxNotifyHandler	518

The IListBoxNotifyHandler class processes events for all classes of list boxes.

This class is designed to handle events that require the list box classes 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IListBoxNotifyHandler

This is the default constructor and accepts no parameters.

```
IListBoxNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IListBoxNotifyHandler

```
virtual
  ~IListBoxNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

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 the list box observers if the following event is received:

- select event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

© IBM Corp. 1992, 1996

IListBoxNotifyHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IListBoxSizeItemEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IListBoxSizeItemEvent

Inherited By None.

Header File ilbdievt.hpp

Members

Member	Page	Member	Page
Constructor	521	setItemSize	522
itemIndex	522	~IListBoxSizeItemEvent	521
itemSize	522		

The IListBoxSizeItemEvent class provides event information for setting the sizes of items in a list box.

Public Functions

Constructors

You can construct and destruct objects of this class.

Although you can construct objects of this class, typically IListBoxDrawItemHandler::dispatchHandlerEvent (p. 516) constructs objects of this class from an object of the class IEvent (p. 304).

IListBoxSizeItemEvent

The only way to construct an object of this class is from an object of the class IEvent (p. 304).

```
IListBoxSizeItemEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

~IListBoxSizeItemEvent

© Copyright IBM Corp. 1992, 1996

IListBoxSizeItemEvent

```
virtual
  ~IListBoxSizeItemEvent();
```

Win	PM	Motif
Y	Y	N

Event Information

Use these members to query and set information about the event. This information is used to identify and size a list box item for drawing.

itemIndex Returns the 0-based index of the item in the list box that needs sizing. This value can be 0 even without any entries in the list box.

```
unsigned long
  itemIndex() const;
```

Win	PM	Motif
Y	Y	N

itemSize Returns the value set by the function setItemSize (p. 522) for the size required to draw the list box item.

```
ISize
  itemSize() const;
```

Win	PM	Motif
Y	Y	N

setItemSize Sets the size required to draw a list box item. All items in the list box are sized to the height specified. The list box also draws all items to the same width, taken from the longest one needed by the items being displayed. Note that the width is only processed if the list box has a horizontal scroll bar.

```
IListBoxSizeItemEvent&
  setItemSize( const ISize& itemSize );
```

Win	PM	Motif
Y	Y	N

Inherited Public Functions

IControlEvent		
controlId		

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

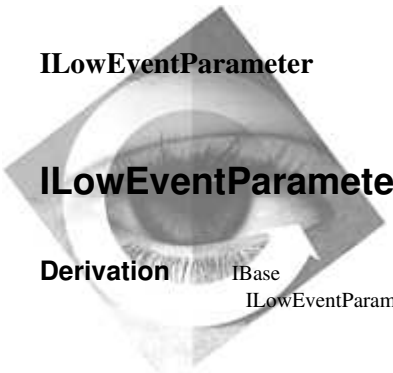
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



ILowEventParameter

Derivation IBase
ILowEventParameter

Inherited By None.

Header File ievtdat2.hpp

The ILowEventParameter class encapsulates the message parameter (MPARAM) and result (MRESULT) data of an event.

ILowEventParameter is an alias of the IEventData class. See IEventData (p. 313) for more information.

In addition, ILowEventParameter is interchangeable with the following classes:

IEventParameter1 (p. 319)
IEventParameter2 (p. 320)
IEventResult (p. 321)
IHighEventParameter (p. 465)

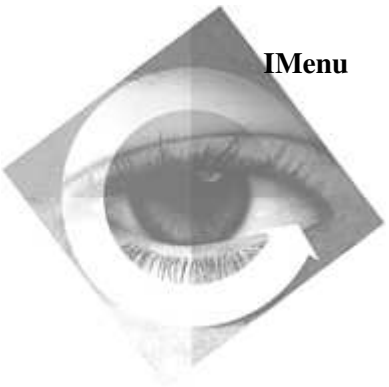
Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



IMenu

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IMenu

Inherited By

IMenuBar
IPopUpMenu
ISubmenu
ISystemMenu

Header File

imenu.hpp

Members

Member	Page	Member	Page
Constructor	536	itemRect	538
add	536	menuHandle	538
addAsNext	536	menuItem	539
addBitmap	527	noStyle	548
addItem	529	numberOfItems	539
addSeparator	529	owner	538
addSubmenu	530	removeConditionalCascade	539
addText	531	removeSubmenu	541
backgroundColor	532	removeSubmenuAt	537
checkItem	543	resetBackgroundColor	533
classDefaultStyle	548	resetDisabledBackgroundColor	534
convertToGUIStyle	546	resetDisabledForegroundColor	534
cursor	536	resetForegroundColor	534
defaultStyle	546	resetHiliteBackgroundColor	534
deleteAt	537	resetHiliteForegroundColor	534
deleteItem	541	selectItem	545
disabledBackgroundColor	532	setBackgroundColor	534
disabledForegroundColor	533	setBitmap	541
disableItem	544	setConditionalCascade	539
elementAt	537	setDefaultStyle	546
enableItem	544	setDisabledBackgroundColor	534
enableNotification	540	setDisabledForegroundColor	535
foregroundColor	533	setForegroundColor	535
hiliteBackgroundColor	533	setHiliteBackgroundColor	535
hiliteForegroundColor	533	setHiliteForegroundColor	535
id	537	setItem	542
isFrameWindow	537	setItemHelpId	540
isItemChecked	545	setSubmenu	542
isItemEnabled	545	setText	543
isValid	538	uncheckItem	546
itemHelpId	538	~IMenu	536

© Copyright IBM Corp. 1992, 1996

IMenu

The IMenu class is a base class for all menu classes, including the following:

- IMenuBar (p. 554)
- IPopUpMenu (p. 712)
- ISubMenu (p. 919)
- ISystemMenu (p. 946)

It provides functions to add, replace, remove, select, and disable menu items in the menu. This class also provides functions to access properties such as the following:

- The color of the menu
- The number of menu items in the menu
- The bounding rectangle for menu items in the menu

PM The bounding rectangle is returned after sending the message MM_QUERYITEMRECT. You can use the bounding rectangle if you need to draw menu items yourself.

Win While IMenu still derives from IWindow, objects of the IMenu (and derived) classes are not windows themselves. This is due to the implementation of menus in the Windows operating system as nonclient areas of the frame. Application code, which relies on the fact that an IMenu is a window in the OS/2 operating system, may not be portable to the Windows operating system.

To support menus loaded from a resource file, IMenu classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Motif IMenu does not support conditional cascading behavior or the drawItem style (drawn buttons) in menus.

The User Interface Class Library provides a Motif callback routine to process the following menu events:

- Showing a menu on the display
- Selecting a menu item in a menu
- Removing a menu from the display

Public Functions

Adding Items

You can add textual, graphical, submenu, or separator items to a menu object. You can also construct an object of the IMenuItem (p. 582) class and add it to the menu object.

Note: Use a value between 1 and 65535 for the item identifier in these functions. Values outside the recommended range can be truncated by the underlying GUI or can result in portability problems. In addition, you need to use a unique value for each menu item within a particular menu or submenu if you want to access items using this class and to identify events resulting from user selection of menu items.

addBitmap

Adds a bitmap menu item as the last item in a menu or submenu.

1

```
virtual IMenu&
  addBitmap( unsigned long newItemId,
             const IBitmapHandle& itemBitmap,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to add a bitmap that you already have loaded into a menu item.

newItemId
The identifier of the new menu item to add to a menu or submenu.

itemBitmap
The bitmap to load for the button face of the new menu item. This parameter accepts a currently available bitmap handle.

intoSubmenuId
The identifier of the submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar must contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

2

```
virtual IMenu&
  addBitmap( unsigned long newItemId,
             const IResourceId& bitmapResId,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to load a bitmap from the resource library you specified when you created the IResourceId object.

© IBM Corp. 1992, 1996

newItemId
The identifier of the new menu item to add to a menu or submenu.

bitmapResId
The resource identifier of the bitmap to load for the button face of the new menu item.

intoSubmenuId
The identifier of the submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar must contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

3
```
virtual IMenu&
  addBitmap( unsigned long newItemId,
             unsigned long bitmapResId,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to load a bitmap from the default resource library.

newItemId
The identifier of the new menu item to add to a menu or submenu.

bitmapResId
The resource identifier of the bitmap to load from the default resource library for the button face of the new menu item.

intoSubmenuId
The identifier of the submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar must contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

addItem

Adds a menu item, represented by an IMenuItem (p. 582), as the last item in a menu or submenu.

menuItem
A new menu item to add to a menu or submenu.

intoSubmenuId
The identifier of a submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

```
virtual IMenu&
  addItem( IMenuItem& menuItem,
           unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar can contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

Exceptions	
IAccessError	The system could not insert the menu item. This may be due to lack of system resources.
IInvalidRequest	The menu item could not be updated with the bitmap.
IInvalidRequest	The bitmap handle is invalid or missing. Verify that the bitmap handle is valid or that the bitmap resource exists.

addSeparator

Adds a separator menu item as the last item in a menu or submenu. Use this function to add items to your menu without creating a menu item object first.

1
```
virtual IMenu&
  addSeparator( unsigned long newItemId,
                unsigned long intoSubmenuId );
```

Win	PM	Motif
Y	Y	Y

Use this version of the function if you need to be able to access the separator item later. Use newItemId to specify the identifier you want to use to access the separator.

newItemId
Item identifier of the new menu item to add to a menu or submenu.

© IBM Corp. 1992, 1996

intosubmenuId
Identifier of the submenu to add the new menu item to.

2
```
virtual IMenu&
  addSeparator( unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to add a separator with an unspecified item identifier.

intosubmenuId
Identifier of the submenu to add the new menu item to.

addSubmenu Creates a submenu and adds it to the menu item.

1
```
virtual IMenu&
  addSubmenu( unsigned long itemId,
              const IResourceId& submenuResId );
```

Win	PM	Motif
Y	Y	Y

itemId
The identifier of the menu item to add the submenu to. The menu item must already be in the menu.

submenuResId
The resource identifier of the submenu to add to the menu item. If you do not specify this parameter, you add a submenu with no entries to the specified menu item.

Win To support menus loaded from a resource file, IMenu classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Exceptions	
IInvalidRequest	An attempt was made to add the menu bar as a submenu. The menu bar cannot be added as a submenu.
IInvalidRequest	A submenu already exists for the menu item. Only one submenu can be added to a menu item.

2
```
virtual IMenu&
  addSubmenu( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

itemId
The identifier of the menu item to add the submenu to. The menu item must already be in the menu. The submenu has no entries.

Exceptions	
IInvalidRequest	An attempt was made to add the menu bar as a submenu. The menu bar cannot be added as a submenu.
IInvalidRequest	A submenu already exists for the menu item. Only one submenu can be added to a menu item.

addText

Adds a text menu item as the last item in a menu or submenu. You can use this function to add items to your menu without creating a menu item object first.

1
```
virtual IMenu&
  addText( unsigned long newItemId,
           const char* itemText,
           unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Use this version to specify a character string for the menu item text.

newItemId
The identifier of the new menu item to add to a menu or submenu.

itemText
The text to load for the button face of the new menu item. This parameter accepts a currently available text string.

intoSubmenuID
The identifier of the submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar must contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

2
```
virtual IMenu&
  addText( unsigned long newItemId,
           const IResourceId& textResId,
           unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

Use this version to load the menu item text from a resource library.

newItemId
The identifier of the new menu item to add to a menu or submenu.

textResId
The text to load for the button face of the new menu item. This parameter accepts the resource ID of a text string to load.

intoSubmenuID
The identifier of the submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif X-Motif provides specific widgets and gadgets for menus. A menu bar must contain only CascadeButton widgets or gadgets because they display a pull-down menu with additional choices. The pull-down menus can contain the following:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

Color

Use color functions to query or change the colors used for menu objects.

backgroundColor

Returns the background color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Win Returns the default background color value of the menu area.

disabledBackgroundColor

Returns the disabled background color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  disabledBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default disabled background color value of the menu area.

disabledForegroundColor

Returns the disabled foreground color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  disabledForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default disabled foreground color value of the menu area.

foregroundColor

Returns the foreground color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Win Returns the default foreground color value of the menu area.

hiliteBackgroundColor

Returns the highlight background color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  hiliteBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default highlight background color value of the menu area.

hiliteForegroundColor

Returns the highlight foreground color value of the menu area or the default if no color for the area has been set.

```
virtual IColor
  hiliteForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default highlight foreground color value of the menu area.

resetBackgroundColor

Resets the background color by undoing a previously set color.

```
virtual IMenu&
  resetBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

© IBM Corp. 1992, 1996

resetDisabledBackgroundColor

Resets the disabled background color by undoing a previously set color.

```
virtual IMenu&
  resetDisabledBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetDisabledForegroundColor

Resets the disabled foreground color by undoing a previously set color.

```
virtual IMenu&
  resetDisabledForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetForegroundColor

Resets the foreground color by undoing a previously set color.

```
virtual IMenu&
  resetForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetHiliteBackgroundColor

Resets the highlight background color by undoing a previously set color.

```
virtual IMenu&
  resetHiliteBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetHiliteForegroundColor

Resets the highlight foreground color by undoing a previously set color.

```
virtual IMenu&
  resetHiliteForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

setBackgroundColor

Sets the background color.

```
virtual IMenu&
  setBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*Y*

setDisabledBackgroundColor

Sets the disabled background color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

```
virtual IMenu&
  setDisabledBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setDisabledForegroundColor

Sets the disabled foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

```
virtual IMenu&
  setDisabledForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setForegroundColor

Sets the foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

```
virtual IMenu&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*Y*

setHiliteBackgroundColor

Sets the highlight background color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

```
virtual IMenu&
  setHiliteBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setHiliteForegroundColor

Sets the highlight foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

```
virtual IMenu&
  setHiliteForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

Constructors

Use these functions to construct and delete objects of the IMenu class. You cannot copy or assign IMenu objects because both the copy constructor and the assignment operator are private functions.

Protected constructors are provided for the use of derived classes to create objects of this class.

© IBM Corp. 1992, 1996

IMenu

IMenu

```
IMenu( const IMenuHandle& menuHandle );
```

Win	PM	Motif
Y	Y	N

Constructs an IMenu object for an existing menu handle. The menu is not automatically destroyed when the IMenu object is destroyed; you can change this by using setAutoDestroyWindow(true) (p. 1068).

~IMenu

```
virtual
  ~IMenu();
```

Win	PM	Motif
Y	Y	Y

Cursored Operations

Cursored operations use objects of the IMenu::Cursor (p. 550) nested class to designate which menu item is to be acted upon. You can use these functions to access each item in the menu without knowing the menu item identifiers.

add

Adds the specified menu item at the specified cursor position by pushing down everything after the cursor.

```
virtual IMenu&
  add( IMenuItem& menuItem,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

addAsNext

Adds the specified menu item as the next item and sets the specified cursor on it.

```
virtual IMenu&
  addAsNext( IMenuItem& menuItem,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

cursor

Returns an object of the nested class IMenu::Cursor (p. 550) and sets the cursor to point to the specified menu item.

itemID

The menu item the cursor is set to point to.

inSubmenuId

The menu or submenu in which the cursor is manipulated. If you specify 0, *itemId* refers to an IMenu object. If you specify a nonzero *itemId*, the cursor is manipulated in the menu or submenu specified by the nonzero *inSubmenuId*.

```
Cursor
  cursor( unsigned long itemId,
          unsigned long inSubmenuId = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

deleteAt Deletes the menu item at the position of the specified cursor and sets the cursor to the previous menu item.

```
virtual IMenu&
  deleteAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

elementAt Returns an IMenuItem that describes the menu item at the cursor position.

```
virtual IMenuItem
  elementAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

removeSubmenuAt

Removes the menu item and submenu indicator at the specified cursor position. Typically, you use this function to remove a cascading menu.

```
virtual IMenu&
  removeSubmenuAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Implementation

These members provide utilities used to implement this class.

id Returns the menu identifier of the menu.

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

Win	PM	Motif
Y	N	N

isFrameWindow

If this object represents a frame window, true is returned. Otherwise, false is returned. Because menus are not frame windows, false is always returned.

```
virtual Boolean
  isFrameWindow() const;
```

Win	PM	Motif
Y	N	N

© IBM Corp. 1992, 1996

isValid If this object represents a valid menu, true is returned. If the menu does not exist, or has already been destroyed, false is returned.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	*N*	*N*

owner Returns the menu's owner. This function may return 0 if either of the following occurs:

- The menu was created without an owner
- The owner window is not represented by an IWindow object

Because a pointer value of 0 can cause unpredictable behavior, check the value of the returned pointer before using it.

```
virtual IWindow*
  owner() const;
```

Win	PM	Motif
Y	*N*	*N*

Item Properties

Menu item properties are general characteristics of a menu item. You can create an IMenuItem (p. 582) object for an item in the menu to manipulate properties of the item. You can also directly obtain certain properties of the menu, such as the number of items contained in it or the rectangle of a particular item.

itemHelpId Returns the identifier of the help panel that is to be displayed when help is requested for the menu item. To display the help panel, you must first associate the window owning this menu with a help instance. See the IHelpWindow (p. 442) class for more information.

```
unsigned long
  itemHelpId( unsigned long menuItemId ) const;
```

Win	PM	Motif
Y	*Y*	*Y*

itemRect Returns the bounding rectangle of a menu item.

```
IRectangle
  itemRect( unsigned long itemId ) const;
```

Win	PM	Motif
Y	*Y*	*Y*

PM The bounding rectangle is the rectangle returned by sending the message MM_QUERYITEMRECT.

menuHandle Returns the handle object for the menu.

```
virtual IMenuHandle
  menuHandle() const;
```

Win	PM	Motif
Y	Y	N

menuItem Returns an IMenuItem (p. 582) object. If the menu does not have a menu item with an ID equal to *itemId*, an exception is thrown.

```
IMenuItem
  menuItem( unsigned long itemId ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

numberOfItems

Returns the number of menu items for the specified submenu. If you specify 0, the count is returned for the IMenu object.

```
unsigned long
  numberOfItems( unsigned long forSubmenuId = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the library queries the resource XmNnumChildren to determine the number of items in a menu or submenu.

removeConditionalCascade

Removes conditional cascade behavior from the specified menu item.

itemWithSubmenuId
The identifier of the menu item from which to remove conditional cascade behavior. If you specify an *itemWithSubmenuId* that does not exist or does not have an associated submenu, an exception is thrown.

```
virtual IMenu&
  removeConditionalCascade( unsigned long itemWithSubmenuId );
```

Win	PM	Motif
Y	Y	I

Motif X-Motif does not support conditionally cascading menus.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

setConditionalCascade

Enables conditionally cascading menus. You can use this function to change the default item on a menu that is already a conditionally cascaded menu. An exception is thrown if either of the following conditions exist:

© IBM Corp. 1992, 1996

- The specified items for *itemWithSubmenuId* or *defaultItemId* do not exist.
- The specified item for *itemWithSubmenuId* does not have a submenu attached.

itemWithSubmenuId
The identifier of the menu item to have its submenu set as a conditionally cascading menu.

defaultItemId
The identifier of the default selection.

```
virtual IMenu&
  setConditionalCascade( unsigned long itemWithSubmenuId,
                         unsigned long defaultItemId );
```

Win	PM	Motif
Y	*Y*	*I*

Motif X-Motif does not support conditionally cascading menus.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

setItemHelpId

Stores the identifier of the help panel that is to be displayed when help is requested for the menu item. To display the help panel, you must first associate the window owning this menu with a help instance. See the IHelpWindow (p. 442) class for more information.

```
virtual IMenu&
  setItemHelpId( unsigned long menuItemId,
                 unsigned long helpTopicId );
```

Win	PM	Motif
Y	*Y*	*Y*

Observer Notification

Observer notification functions implement the public INotifier protocol for IMenu classes.

enableNotification

Enables the menu to send notifications to any observer objects.

```
virtual IMenu&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*Y*

Removing Items

You can remove menu items or submenus by providing the item identifier of the item to remove.

deleteItem Deletes the item with the specified identifier from the menu.

```
virtual IMenu&
  deleteItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

removeSubmenu

Removes the menu item's submenu indicator and destroys the submenu.

itemWithSubmenuId
The identifier of the item whose submenu is to be removed.

```
virtual IMenu&
  removeSubmenu( unsigned long itemWithSubmenuId );
```

Win	PM	Motif
Y	Y	Y

Replacing Items

You replace a menu item to change the text or graphic displayed on the button face or to create a submenu. You can also replace a menu item with an IMenuItem (p. 582) object. It is often convenient to create an IMenuItem object using menuItem (p. 539), make the appropriate changes to the IMenuItem object, and then replace the item in the menu object using setItem (p. 542).

setBitmap Causes a menu item to display a bitmap.

1
```
virtual IMenu&
  setBitmap( unsigned long menuItemId,
             unsigned long newBitmapResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want User Interface Class Library to load the bitmap from the default resource library.

menuItemId
The identifier of the menu item to add the bitmap to.

newBitmapResId
The bitmap to display. This parameter accepts the resource identifier of a bit map in the default resource library.

2
```
virtual IMenu&
  setBitmap( unsigned long menuItemId,
             const IBitmapHandle& bitmapHandle );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Use this function when you already have the bitmap loaded.

menuItemId
The identifier of the menu item to add the bitmap to.

bitmapHandle
The handle of the bitmap to display.

3
```
virtual IMenu&
  setBitmap( unsigned long menuItemId,
             const IResourceId& newBitmapResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want User Interface Class Library to load the bitmap from the resource library you specified when you created the IResourceId object.

menuItemId
The identifier of the menu item to add the bitmap to.

newBitmapResId
The bitmap to display. This parameter accepts the resource identifier of a bitmap.

setItem

Replaces a specified menu item's style and representation.

```
virtual IMenu&
  setItem( const IMenuItem& menuItem );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The system could not insert the menu item. This may be due to lack of system resources.
IInvalidRequest	The menu item could not be updated with the bitmap.
IInvalidRequest	The bitmap handle is invalid or missing. Verify that the bitmap handle is valid or that the bitmap resource exists.

setSubmenu

Creates a menu and sets it as the menu item's submenu.

itemId
The menu item ID that causes a submenu to be displayed when it is selected. The menu item must already exist.

submenuResId
The resource ID of the submenu to be set.

```
virtual IMenu&
  setSubmenu( unsigned long itemId,
              const IResourceId& submenuResId );
```

Win	PM	Motif
Y	Y	Y

Win To support menus loaded from a resource file, IMenu classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Exceptions	
IInvalidRequest	An attempt was made to add the menu bar as a submenu. The menu bar cannot be added as a submenu.

setText Replaces a specified menu item's text.

menuItemId
ID of the menu item whose text will be set.

newText
New text for the menu item.

newTextResId
Resource ID of the new text for the menu item.

1
```
virtual IMenu&
  setText( unsigned long menuItemId,
           const char* newText );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The system could not insert the menu item. This may be due to lack of system resources.

2
```
virtual IMenu&
  setText( unsigned long menuItemId,
           const IResourceId& newTextResId );
```

Win	PM	Motif
Y	Y	Y

Selection

Selection functions allow you to determine if a menu item is selectable (enabled) or is displaying a selection state indicator. You can also change these properties.

checkItem Places a select state indicator to the left of the specified item. The default visual of this indicator is a check mark.

© IBM Corp. 1992, 1996

itemId
The identifier of the menu item to add or remove the indicator from.

check
An optional Boolean parameter that specifies whether the menu item indicator should be added or removed.

Note: You cannot use this function on the menu bar.

```
virtual IMenu&
  checkItem( unsigned long itemId,
             Boolean check = true );
```

Win	PM	Motif
Y	Y	Y

PM The select state indicator is a check mark to the left of the specified item.

Motif The select state indicator is a square object to the left of the specified item.

In Motif, the resources XmNset and XmNindicatorOn are set to check or uncheck a menu item.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.
IInvalidRequest	The system could not insert the menu item. This may be due to lack of system resources.

disableItem Prevents the menu item with the specified identifier from being selected.

```
virtual IMenu&
  disableItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the resource XmNsensitive is set by invoking XtSetSensitive to enable or disable a menu item.

enableItem Makes a menu item selectable.

itemId
The identifier of the menu item to be enabled or disabled.

enable
Specifies whether the menu item should be enabled. Optional.

```
virtual IMenu&
  enableItem( unsigned long itemId,
              Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the resource XmNsensitive is set by invoking XtSetSensitive to enable or disable a menu item.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.
IInvalidRequest	The system could not insert the menu item. This may be due to lack of system resources.

isItemChecked

If the menu item with the specified identifier is checked, true is returned. Otherwise, false is returned.

```
Boolean
  isItemChecked( unsigned long itemId ) const;
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the resources XmNset and XmNindicatorOn are queried to determine if a menu item is checked.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

isItemEnabled

If the menu item with the specified identifier is selectable, true is returned. Otherwise, false is returned.

```
Boolean
  isItemEnabled( unsigned long itemId ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

selectItem

Selects the menu item with the specified identifier.

```
virtual IMenu&
  selectItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the resource XmNset is set to select a menu item.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

uncheckItem Removes the select state indicator from the menu item with the specified identifier.

```
virtual IMenu&
  uncheckItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the resources XmNset and XmNindicatorOn are set to check or uncheck a menu item.

Styles

IMenu defines objects of the nested class IMenu::Style (p. 553). You can use these styles with the function IMenu::setDefaultStyle (p. 546).

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles that are defined by the User Interface Class Library can be returned by setting the *extendedOnly* parameter to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is IMenu::classDefaultStyle (p. 548) unless you have changed it using IMenu::setDefaultStyle (p. 546).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent menus.

```
static void
  setDefaultStyle( const Style& aStyle );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Constructors

Use these functions to construct and delete objects of the IMenu class. You cannot copy or assign IMenu objects because both the copy constructor and the assignment operator are private functions.

Protected constructors are provided for the use of derived classes to create objects of this class.

IMenu

```
IMenu();
```

Win	PM	Motif
Y	*Y*	*Y*

Provided for derived classes to call.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Public Data

Styles

IMenu defines objects of the nested class IMenu::Style (p. 553). You can use these styles with the function IMenu::setDefaultStyle (p. 546).

classDefaultStyle

Specifies the original default style for this class, which is IMenu::noStyle (p. 548).

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

noStyle

Specifies that no style applies to the menu.

```
static const Style
  noStyle;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

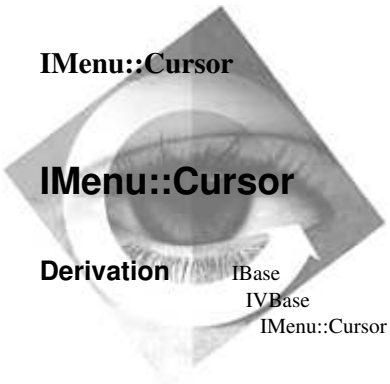
IBase		
recoverable	unrecoverable	

Nested Classes

IMenu contains the following nested classes:

IMenu::Cursor (see page 550)
IMenu::Style (see page 553)

© IBM Corp. 1992, 1996



IMenu::Cursor

Derivation

IBase
IVBase
IMenu::Cursor

Inherited By None.

Header File imenu.hpp

Members

Member	Page	Member	Page
Constructor	550	setToLast	551
Cursor	550	setToNext	551
invalidate	551	setToPrevious	551
isValid	551	~Cursor	551
setToFirst	551		

The IMenu::Cursor class creates and manages the cursor for menu items. You can add or update items anywhere in a menu structure using this class.

Public Functions

Constructors

You can only construct objects of this class from an object of the IMenu (p. 525) class and a submenu ID. The submenu defaults to 0 for the top-level menu items. You can also destruct objects of this class.

Cursor Use this function to construct a cursor object for a menu.

```
Cursor( const IMenu& menu,
        unsigned long forSubmenuId = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

You can define a cursor for a submenu using *forSubmenuId* to specify the identifier of the submenu.

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.
IInvalidRequest	The menu is not valid. A valid menu object must be provided.

© Copyright IBM Corp. 1992, 1996

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Cursor Movement

Use cursor movement members to move the cursor across the IMenu object.

setToFirst Points to the first menu item. If successful, true is returned.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToLast Points to the last menu item. If successful, true is returned.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Points to the next menu item. If there is none, the cursor is invalidated. If successful, true is returned.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Points to the previous menu item. If there is none, the cursor is invalidated. If successful, true is returned.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation

Use cursor validation members to determine if the cursor is valid or to invalidate it. The cursor must be valid in order to use it to access an item.

invalidate Flags the cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Queries whether this cursor points to a valid menu item.

© IBM Corp. 1992, 1996

IMenu::Cursor

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IMenu::Style

Derivation

IBase
  IBitFlag
    IMenu::Style

Inherited By None.

Header File imenu.hpp

The nested class IMenu::Style provides a set of valid styles for the IMenu (p. 525) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IMenuBar

Derivation

IBase
IVBase
INotifier
IWindow
IMenu
IMenuBar

Inherited By None.

Header File imenubar.hpp

Members

Member	Page	Member	Page
Constructor	555	setDefaultStyle	556
classDefaultStyle	558	setMenu	556
convertToGUIStyle	556	unregisterCallbacks	558
defaultStyle	556	wrapper	559
empty	559	~IMenuBar	555
registerCallbacks	558		

The IMenuBar class represents a menu bar for frame windows.

Note: You can add a menu bar to a frame window without using the IMenuBar class by specifying the style IFrameWindow::menuBar (p. 386) when you construct an IFrameWindow (p. 349) object.

Win To support menus loaded from a resource file, IMenuBar classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Motif The User Interface Class Library adds a Motif callback routine to all menu bars for processing selected menu bar items.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IMenuBar objects because the copy constructor and assignment operator are private functions.

© Copyright IBM Corp. 1992, 1996

IMenuBar

1 `IMenuBar( const IResourceId& menuResId,`
`IFrameWindow* owner );`

Win	PM	Motif
Y	Y	Y

Creates a menu bar with a menu resource ID and a frame window owner. Use this constructor to create a menu bar for a frame window with a menu resource from your resource library.

Win To support menus loaded from a resource file, IMenuBar classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Exceptions	
IInvalidParameter	*owner* is 0. You must specify a valid IFrameWindow as the owner for the menu bar.

2 `IMenuBar( IFrameWindow* owner,`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	Y	Y

Creates an empty menu bar or a wrapper for an existing IFrameWindow (p. 349) menu bar. Use this constructor to create the following:

- A menu bar with no entries, using the style IMenuBar::empty (p. 559)
- A menu bar wrapper, using the style IMenuBar::wrapper (p. 559)

Exceptions	
IInvalidParameter	*owner* is 0. You must specify a valid IFrameWindow as the owner for the menu bar.
IInvalidRequest	The style specified is invalid. Specify one of the styles IMenuBar::wrapper or IMenuBar::empty, but not both.
IInvalidRequest	The style IMenuBar::wrapper was specified, but the frame window has no menu bar. Verify that the IFrameWindow::menuBar style was specified when the frame was created and that *owner* is the correct IFrameWindow.

~IMenuBar

```
virtual
  ~IMenuBar();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Replacing Menus

You can replace the menu currently being used on the menu bar with another menu.

setMenu Sets or changes the menu bar to the specified menu resource ID.

```
virtual IMenuBar&
  setMenu( const IResourceId& menuResId );
```

Win	PM	Motif
Y	Y	Y

Win To support menus loaded from a resource file, IMenuBar classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Styles

IMenuBar defines objects of the nested class IMenuBar::Style (p. 561). You can use these styles with the function IMenuBar::setDefaultStyle (p. 556) and a constructor for IMenuBar.

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is IMenuBar::classDefaultStyle (p. 558) unless you have changed it using IMenuBar::setDefaultStyle (p. 556).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent menu bars.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IMenu		
add	enableNotification	resetDisabledBackgroundColor
addAsNext	foregroundColor	resetDisabledForegroundColor
addBitmap	hiliteBackgroundColor	resetForegroundColor
addItem	hiliteForegroundColor	resetHiliteBackgroundColor
addSeparator	id	resetHiliteForegroundColor
addSubmenu	isFrameWindow	selectItem
addText	isItemChecked	setBackgroundColor
backgroundColor	isItemEnabled	setBitmap
checkItem	isValid	setConditionalCascade
convertToGUIStyle	itemHelpId	**setDefaultStyle**
cursor	itemRect	setDisabledBackgroundColor
defaultStyle	menuHandle	setDisabledForegroundColor
deleteAt	menuItem	setForegroundColor
deleteItem	numberOfItems	setHiliteBackgroundColor
disabledBackgroundColor	owner	setHiliteForegroundColor
disabledForegroundColor	removeConditionalCascade	setItem
disableItem	removeSubmenu	setItemHelpId
elementAt	removeSubmenuAt	setSubmenu
enableItem	resetBackgroundColor	setText

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Adds X-Motif callbacks for an XmMenuBar row column widget.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes X-Motif callbacks from a XmMenuBar row column widget.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

IMenuBar defines objects of the nested class IMenuBar::Style (p. 561). You can use these styles with the function IMenuBar::setDefaultStyle (p. 556) and a constructor for IMenuBar.

classDefaultStyle

Specifies the original default style for this class, which is IMenuBar::empty (p. 559).

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

empty Creates a menu with no items. You can dynamically populate the menu using functions from IMenu (p. 525) and ISubmenu (p. 919). You cannot specify this style with the IMenuBar::wrapper (p. 559) style.

```
static const Style
  empty;
```

Win	PM	Motif
Y	*Y*	*Y*

wrapper Specifies that the menu bar is a wrapper for an IFrameWindow (p. 349) menu bar, which you previously created using the style IFrameWindow::menuBar (p. 386). You cannot specify this style with the IMenuBar::empty (p. 559) style.

```
static const Style
  wrapper;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IMenu		
classDefaultStyle	**noStyle**	

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IMenuBar

Nested Classes

IMenuBar contains the following nested classes:

IMenuBar::Style (see page 561)

IMenuBar::Style

Derivation

IBase
  IBitFlag
    IMenuBar::Style

Inherited By None.

Header File imenubar.hpp

The nested class IMenuBar::Style represents creation flags for the IMenuBar (p. 554) class. IMenuBar defines IMenuBar::Style objects (p. 558) that you pass to an IMenuBar constructor.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IMenuDrawItemEvent

Derivation

IBase
IVBase
IEvent
IControlEvent
IDrawItemEvent
IMenuDrawItemEvent

Inherited By None.

Header File imndievt.hpp

Members

Member	Page	Member	Page
Constructor	562	isSelected	563
isChecked	563	~IMenuDrawItemEvent	562
isDisabled	563		

The IMenuDrawItemEvent class provides the draw-item event information for painting draw-item menu items.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMenuDrawItemEvent

Constructs an IMenuDrawItemEvent object from the specified event. IMenuDrawItemHandler::dispatchHandlerEvent (p. 567) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to its event-processing functions.

```
IMenuDrawItemEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Creates an IMenuDrawItemEvent object from an IEvent (p. 304) object.

~IMenuDrawItemEvent

© Copyright IBM Corp. 1992, 1996

```
virtual
  ~IMenuDrawItemEvent();
```

Win	PM	Motif
Y	Y	Y

Menu Item Attributes

Use menu item attribute members to determine the display attributes of a menu item about to be drawn. You only need to use these functions if you do your own drawing for the selected, checked, or disabled display attributes of a menu item. Call these members only from within an overridden version of IMenuDrawItemHandler::draw (p. 567).

isChecked If the menu item is checked, true is returned.

```
Boolean
  isChecked() const;
```

Win	PM	Motif
Y	Y	Y

isDisabled If the menu item is disabled, true is returned.

```
Boolean
  isDisabled() const;
```

Win	PM	Motif
Y	Y	Y

isSelected If the menu item is selected, true is returned.

```
Boolean
  isSelected() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IDrawItemEvent		
itemId	itemPresSpaceHandle	itemRect

IControlEvent		
controlId		

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

© IBM Corp. 1992, 1996

IMenuDrawItemEvent

IEvent		
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

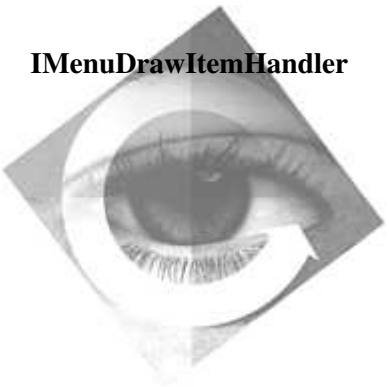
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IDrawItemEvent		
ownerItemData		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	



IMenuDrawItemHandler

Derivation

IBase
  IVBase
    IHandler
      IMenuDrawItemHandler

Inherited By None.

Header File imndihdr.hpp

Members

Member	Page	Member	Page
Constructor	566	drewSelected	569
dispatchHandlerEvent	567	highlight	568
draw	567	setSize	568
drewChecked	569	unhighlight	568
drewDisabled	569	~IMenuDrawItemHandler	566

The IMenuDrawItemHandler class processes menu draw-item events.

Note: You must create the menu item with the constructor from IMenuItem (p. 582) that creates a draw-item. You can also call IMenuItem::setDrawItem (p. 590) after you have created the menu item.

Create a handler derived from IMenuDrawItemHandler and attach it to a menu. Do this by calling IHandler::handleEventsFor (p. 413) to pass the appropriate menu to the menu draw-item handler.

When the menu draw-item handler receives a menu draw-item event, it creates an IMenuDrawItemEvent (p. 562) object and routes that object to the appropriate IMenuDrawItemHandler virtual function. You can override these virtual functions to supply your own specialized processing of a menu draw-item event.

The return value from the virtual functions specifies whether the menu draw-item event should be passed on for additional processing, as follows:

Return Value	Meaning
true	The menu draw-item event requires no additional processing. Do not pass it to another handler.
false	The menu draw-item event may require additional processing. Pass the menu draw-item event to the next handler for additional processing, as follows:

© Copyright IBM Corp. 1992, 1996

- If there is another handler for the menu, pass the menu draw-item event to the next handler.
- If this is the last handler for the menu, call IWindow::dispatch (p. 1082) to dispatch the menu draw-item event to the menu's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the menu draw-item event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMenuDrawItemHandler

Provides the default constructor.

```
IMenuDrawItemHandler();
```

Win	PM	Motif
Y	Y	Y

~IMenuDrawItemHandler

```
virtual
  ~IMenuDrawItemHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**

IBase		
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IMenuDrawItemHandler object processes only drawing events for menu items.

dispatchHandlerEvent

If a menu draw-item event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

Event-processing functions are called when an event occurs that may need to be processed by this handler.

draw

By default, the system draws a selected indicator, creates a halftone of the text for a disabled menu item, and inverts the bits to highlight menu item selection. Unless the system default method is unacceptable, you do not have to draw these attributes.

Override this function to draw the menu-item. If you draw any of these attributes, set the appropriate IMenuDrawItemHandler::DrawFlag (p. 570) before returning from IMenuDrawItemHandler::draw. This prevents the system from drawing these attributes.

The class IMenuDrawItemEvent (p. 562) provides menu item attribute functions to query whether the menu item is selected, checked, or disabled.

```
virtual Boolean
  draw( IMenuDrawItemEvent& event,
        DrawFlag& flag );
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

highlight Highlights a selected menu item by inverting the bits.

Override this function if you want to draw the highlight that indicates a menu item has been selected. Do this only if the default of inverting the bits for a selected item is unacceptable.

Note: This function is not called if draw (p. 567) returns true.

```
virtual Boolean
  highlight( IMenuDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

setSize Override this function to set the width and height of each draw item menu item. This event is dispatched only once for each menu item, when the menu is initialized. Specify the size in *newSize*.

```
virtual Boolean
  setSize( IMenuDrawItemEvent& event,
           ISize& newSize );
```

Win	PM	Motif
Y	Y	Y

unhighlight Removes the highlight from a deselected menu item.

Override this function if you want to remove the highlight that indicates a menu item has been deselected. Do this only if the default is unacceptable.

Note: This function is not called if draw (p. 567) returns true.

```
virtual Boolean
  unhighlight( IMenuDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Protected Data

Drawing Flags

An object of the nested class IMenuDrawItemHandler::DrawFlag (p. 570) is passed as the second argument of an overridden version of IMenuDrawItemHandler::draw (p. 567). The overridden version must set the appropriate drawing flags to indicate whether it handled any of the special drawing needs of the menu item.

drewChecked

Indicates that the selected indicator of the menu item has been drawn. If you draw the selected indicator in your override of IMenuDrawItemHander::draw (p. 567), bitwise OR (|) this value into the second parameter of the function to prevent the default drawing from occurring.

```
static const DrawFlag
  drewChecked;
```

Win	PM	Motif
Y	Y	Y

drewDisabled

Indicates that the menu item's disabled state has been drawn. If you draw the disabled emphasis in your override of IMenuDrawItemHander::draw (p. 567), bitwise OR (|) this value into the second parameter of the function to prevent the default drawing from occurring.

```
static const DrawFlag
  drewDisabled;
```

Win	PM	Motif
Y	Y	Y

drewSelected

Indicates that the highlighting of the menu item has been drawn. If you draw the selected emphasis highlighting in your override of IMenuDrawItemHander::draw (p. 567), bitwise OR (|) this value into the second parameter of the function to prevent the default drawing from occurring.

```
static const DrawFlag
  drewSelected;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IMenuDrawItemHandler contains the following nested classes:

IMenuDrawItemHandler::DrawFlag (see page 570)

© IBM Corp. 1992, 1996

IMenuDrawItemHandler::DrawFlag

Derivation IBase
IBitFlag
IMenuDrawItemHandler::DrawFlag

Inherited By None.

Header File imndihdr.hpp

The nested class IMenuDrawItemHandler::DrawFlag provides flags used by the function IMenuDrawItemHandler::draw (p. 567). You can combine objects of this class using the bitwise OR (|) operator. Objects of this class are declared in the class IMenuDrawItemHandler (p. 568).

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IMenuEvent

Derivation

IBase
  IVBase
    IEvent
      IMenuEvent

Inherited By None.

Header File imenuevt.hpp

Members

Member	Page	Member	Page
Constructor	571	mousePosition	572
menuItem	572	~IMenuEvent	571
menuItemId	572		

The IMenuEvent class provides information about a menu event for the event-handling functions of IMenuHandler (p. 576).

Public Functions

Constructors

You can construct and destruct objects of this class.

IMenuEvent Constructs an IMenuEvent object from the specified event. IMenuHandler::dispatchHandlerEvent (p. 578) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to its virtual functions.

```
IMenuEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IMenuEvent

```
virtual
  ~IMenuEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Event Information

These members query the menu event information of instances of this class.

menuItem Returns an object of IMenuItem (p. 582) for the menu item. This function only returns an actual IMenuItem when the IMenuEvent resulted from the user selecting a menu item. If this function is called during any of the other menu actions, an IMenuItem representing an empty *itemId* 0 is returned.

```
IMenuItem
  menuItem() const;
```

Win	PM	Motif
Y	Y	Y

menuItemId Returns the ID of the menu item. The returned menu item ID can be operated upon by IMenuHandler::menuEnded (p. 579), IMenuHandler::menuSelected (p. 579), and IMenuHandler::menuShowing (p. 579).

```
unsigned long
  menuItemId() const;
```

Win	PM	Motif
Y	Y	Y

mousePosition

Returns the mouse position at the time the menu event occurred. You can use the value returned as the parameter on IPopUpMenu::show (p. 714).

```
IPoint
  mousePosition() const;
```

Win	PM	Motif
Y	Y	Y

PM In Presentation Manager, the User Interface Class Library maps the mouse position to the menu's owner window's coordinates.

Motif In Motif, the User Interface Class Library maps the mouse position to the coordinates of the IWindow over which the mouse click occurred.

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**	

© IBM Corp. 1992, 1996

IMenuHandle

Derivation

IBase
IHandle
IMenuHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	574

The IMenuHandle class accesses operating system menus.

Win IMenuHandle is an alias for the Windows Programmer's Toolkit type HMENU.

Public Functions

Constructors

You can construct objects of this class.

IMenuHandle Constructs objects of this class from a menu handle (a value of type IHandle::Value), which defaults to 0.

```
IMenuHandle( Value hmenu = 0 );
```

Win	PM	Motif
Y	*N*	*N*

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

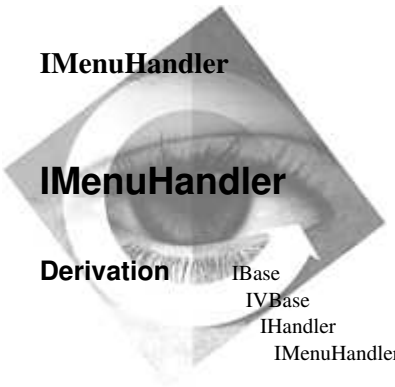
© Copyright IBM Corp. 1992, 1996

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IMenuHandler

Derivation

IBase
IVBase
IHandler
IMenuHandler

Inherited By ICnrMenuHandler

Header File imenuhdr.hpp

Members

Member	Page	Member	Page
Constructor	577	menuSelected	579
addSourceEmphasis	580	menuShowing	579
dispatchHandlerEvent	578	removeSourceEmphasis	580
makePopUpMenu	578	~IMenuHandler	577
menuEnded	579		

The IMenuHandler class processes all types of menu events except menu command events, which are handled by the class ICommandHandler (p. 214). You can override IMenuHandler::menuShowing (p. 579) to make changes via the ISubmenu (p. 919) object that is passed as a parameter. You can also use this class to display a pop-up menu. If you want to display pop-up menus for IContainerObject (Vol. III) or IContainerControl (Vol. III) objects, use the ICnrMenuHandler-derived (Vol. III) class.

Create a handler derived from IMenuHandler and attach it to a frame window for a regular menu or to a control for a pop-up menu. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the appropriate frame window or control to the menu handler.

When the menu handler receives a menu event, it creates an IMenuEvent (p. 571) object and routes that object to the appropriate IMenuHandler virtual functions. Override these virtual functions to supply your own specialized processing of a menu event.

The return value from the virtual functions specifies whether the menu event is passed on for additional processing, as follows:

true The menu event requires no additional processing. Do not pass it to another handler.

© Copyright IBM Corp. 1992, 1996

false The menu event may require additional processing. Pass the menu event to the next handler for additional processing, as follows:

- If there is another handler for the frame window or control, pass the menu event to the next handler.
- If this is the last handler for the frame window or control, call IWindow::defaultProcedure (p. 1082) to process the menu event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMenuHandler

Default constructor.

```
IMenuHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

~IMenuHandler

```
virtual
  ~IMenuHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IMenuHandler object processes only menu-related events.

dispatchHandlerEvent

If a menu event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members are called when an event occurs that may need to be processed by this handler.

makePopUpMenu

Shows a pop-up menu. It is called when the user causes a context menu event to occur while the cursor is on top of a control associated with the IMenuHandler. A context menu event is associated with a context or pop-up menu. Within makePopUpMenu, you can do either of the following:

- Create and show a pop-up menu
- Show an existing pop-up menu

You can override this function to create and show a pop-up menu by using the class IPopUpMenu (p. 712). Code your override function to return true if your program shows an IPopUpMenu; otherwise, code it to return false.

The user has dismissed the pop-up menu when IMenuHandler::menuEnded (p. 579) is called.

The storage for the IPopUpMenu object menu can be allocated from the following:

- Available free storage, if you use the global operator new
- The stack
- Statically

If you allocate the IPopUpMenu using the global operator new, you can create the menu object within makePopUpMenu and request that the User Interface Class Library delete your IPopUpMenu object automatically when the menu ends. To do

this, call IWindow::setAutoDeleteObject (p. 1068) for the IPopUpMenu after creating it. Your implementation of makePopUpMenu should then call IPopUpMenu::show (p. 714) and return true.

If you allocate the IPopUpMenu on the stack or statically, the allocation should be done outside of makePopUpMenu. In this case, makePopUpMenu only needs to show the menu and return true.

```
virtual Boolean
  makePopUpMenu( IMenuEvent& menuEvent );
```

Win	PM	Motif
Y	Y	Y

menuEnded

Processes a menu end event. It is called when a submenu, such as a pull-down menu, cascading menu, or pop-up menu ends.

```
virtual Boolean
  menuEnded( IMenuEvent& menuEvent );
```

Win	PM	Motif
Y	Y	Y

menuSelected

Called when a menu item is selected.

This function or, if you have one, a derived implementation of this function, is called when a menu item (menu bar, pull-down menu, or cascade menu) is highlighted. A user can highlight a menu item by doing any of the following:

- Using arrow keys
- Moving the mouse pointer over the menu item while pressing the select button (typically, mouse button 1)
- Using the Alt or F10 keys

  Note: These keys only highlight the first item on your menu bar and cause this function to be called.

The class IInfoArea (Vol. III) uses this function to determine the information text to display based on the highlighted menu item.

```
virtual Boolean
  menuSelected( IMenuEvent& menuEvent );
```

Win	PM	Motif
Y	Y	Y

menuShowing

Called before a submenu, such as a pull-down, cascading, or pop-up menu is shown.

© IBM Corp. 1992, 1996

```
virtual Boolean
  menuShowing( IMenuEvent& menuEvent,
               ISubmenu& submenuAboutToShow );
```

Win	PM	Motif
Y	Y	Y

menuEvent
The current menu event object.

submenuAboutToShow
A submenu object to temporarily change the menu items.

The *submenuAboutToShow* argument of this function is an object of the ISubmenu (p. 919) class representing the submenu that is about to be shown. You can override menuShowing in your IMenuHandler derived class and use the ISubmenu object to alter the contents of the submenu temporarily and to cause the altered submenu to be shown. If you alter the submenu object, you must return true from the menuShowing function. If menuShowing returns true, any change to the ISubmenu is undone when IMenuHandler::menuEnded (p. 579) is called after the menu terminates. This allows you to account for situations in your application where you may want to dynamically alter menu choices. An example of this is disabling menu items that cannot be selected at this time.

The default menuShowing function provided with IMenuHandler does not alter the submenu and returns false.

Menu Emphasis

You may find it useful to give a special display emphasis to the user-interface element to which a menu applies. For example, if the user displays a pop-up menu for a list box, you could change the look of the list box items that would be affected by a selection from the pop-up.

addSourceEmphasis

Calls IWindow::showSourceEmphasis (p. 1077). Although the IWindow implementation of showSourceEmphasis does nothing, some derived classes, such as IContainerControl (Vol. III), perform specialized processing for source emphasis.

```
virtual void
  addSourceEmphasis( const IMenuEvent& menuEvent );
```

Win	PM	Motif
Y	Y	Y

removeSourceEmphasis

Calls IWindow::hideSourceEmphasis (p. 1075).

```
virtual void
  removeSourceEmphasis( const IMenuEvent& menuEvent );
```

Win	PM	Motif
Y	Y	Y

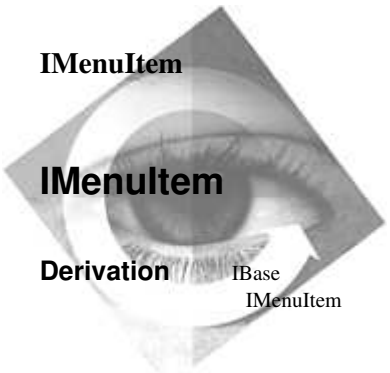
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IMenuItem

Derivation IBase
IMenuItem

Inherited By None.

Header File imnitem.hpp

Members

Member	Page	Member	Page
Constructor	585	noDismiss	594
atEnd	594	noStyle	595
attribute	592	operator =	585
bitmap	587	postHelp	595
buttonSeparator	595	postSystemCommand	596
checked	593	separator	596
classDefaultAttribute	594	setAttribute	592
classDefaultStyle	595	setBitmap	587
commandType	584	setChecked	586
convertToGUIStyle	590	setCommand	584
defaultAttribute	584	setDefaultAttribute	584
defaultStyle	591	setDefaultStyle	591
disabled	594	setDisabled	586
drawItem	595	setDrawItem	590
extendedStyle	592	setExtendedStyle	592
framed	594	setFramed	586
helpId	592	setHelpId	593
highlighted	594	setHighlighted	587
id	589	setIndex	589
index	589	setLayout	589
isBitmap	587	setNoDismiss	587
isChecked	585	setSelectable	590
isDisabled	585	setSeparator	590
isDrawItem	589	setStyle	593
isFramed	586	setSubmenuHandle	591
isHighlighted	586	setText	588
isNoDismiss	586	split	596
isSelectable	589	splitWithSeparator	596
isSeparator	590	style	593
isSubmenu	591	submenuHandle	591
isText	587	text	588
layoutType	589	unavailable	597
noAttribute	594	~IMenuItem	585

The IMenuItem class represents properties of a menu item. This class, which is a data class, provides the following:

© Copyright IBM Corp. 1992, 1996

- Style and attribute objects to manipulate the menu item's appearance or behavior
- Functions to set the label text or bitmap
- Functions to query the label text or bitmap

These functions do not affect the menu unless you call IMenu::addItem (p. 529), IMenu::setItem (p. 542), ISubmenu::addItem (p. 921), or ISubmenu::setItem (p. 927) to place the menu item object into the menu.

You can construct an IMenuItem directly, set up the properties you want on it, and then place it in the menu using the IMenu or ISubmenu member functions. This is useful when you are creating new menu items.

Alternatively, you can use IMenu::menuItem (p. 539) to create an IMenuItem object. You can change this object as needed and then place it back into the menu using the IMenu or ISubmenu functions to have the changes take effect. This is useful when you want to change the properties of existing menu items.

You can use menu items to generate the following events:

- command
- systemCommand
- helpCommand

Note: ICommandEvents do not specify whether they are for a menu bar or pop-up menu. If you need to tell the difference, you must use different IDs for the menu items to distinguish whether they are on the menu bar or the pop-up menu. For example, you might need to apply different actions to a menu item object when the user selects it from the menu bar as opposed to the pop-up menu.

PM A menu item identifier does not have to be unique. However, if you call a function such as IMenu::checkItem (p. 543) using the identifier, the first menu item that has the identifier is acted upon.

Motif All menu item IDs must be unique.

Public Functions

Attributes

For a menu item, *styles* represent properties of the menu item that typically do not change during the life of the item. *Attributes* represent properties that are more likely to be changed by the application during the life of the menu item. All attributes are set independently of each other.

© IBM Corp. 1992, 1996

IMenuItem defines a set of objects of the nested class IMenuItem::Attribute (p. 599). You can use these attributes with constructors for IMenuItem, and IMenuItem::setDefaultAttribute (p. 584).

defaultAttribute

Returns the default attribute. The default attribute is classDefaultAttribute (p. 594) unless you have changed it using setDefaultAttribute (p. 584).

```
static Attribute
  defaultAttribute();
```

Win	PM	Motif
Y	Y	Y

setDefaultAttribute

Sets the default attribute for all subsequent menu items.

```
static void
  setDefaultAttribute( const Attribute& attribute );
```

Win	PM	Motif
Y	Y	Y

Command Type

You can specify the type of event that is generated when a menu item is selected. Use these members to query the event type that is generated by a menu item or to set a new type of event.

commandType

Returns the event the menu item generates when selected. The event can be a command, system command, or help command.

```
CommandType
  commandType() const;
```

Win	PM	Motif
Y	Y	Y

setCommand

Specifies what event to generate when the user selects the menu item. The event can be a command, system command, or help command.

```
IMenuItem&
  setCommand( CommandType value );
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

In most cases you need to call at least one of the set functions on the menu item before it can be added to a menu. For example, if you want a bitmap menu item, first construct the IMenuItem, and then call setBitmap (p. 587) to specify the bitmap to be used for display purposes.

After construction, you can add IMenuItems that have the styles IMenuItem::separator (p. 596) or IMenuItem::drawItem (p. 595) to a menu without having to call any of the set functions.

You can create IMenuItem objects from an existing menu item by calling IMenu::menuItem (p. 539). This is useful if you want to modify a menu item that already exists in a menu.

IMenuItem Creates an IMenuItem object with the specified item identifier, styles, and attributes.

Note: Use a value between 1 and 65565 for the item identifier. Values outside the recommended range may be truncated by the underlying GUI or result in portability problems. In addition, use a unique value for each menu item within a particular menu or submenu. You need unique menu item identifiers if you want to access items using the IMenu (p. 525) or ISubmenu (p. 919) classes and to identify events resulting from user selection of menu items.

1
```
IMenuItem( const IMenuItem& menuItem );
```

Win	PM	Motif
Y	Y	Y

2
```
IMenuItem( unsigned long itemId,
           const Style& style = defaultStyle ( ),
           const Attribute& attribute = defaultAttribute ( ) );
```

Win	PM	Motif
Y	Y	Y

operator = Assigns the value of one menu item object to another.

```
IMenuItem&
  operator =( const IMenuItem& menuItem );
```

Win	PM	Motif
Y	Y	Y

~IMenuItem

```
~IMenuItem();
```

Win	PM	Motif
Y	Y	Y

Item State

You can use attribute functions to query or change menu item attributes.

isChecked If the attribute checked (p. 593) is set, true is returned.

```
Boolean
  isChecked() const;
```

Win	PM	Motif
Y	Y	Y

isDisabled If the attribute disabled (p. 594) is set, true is returned.

© IBM Corp. 1992, 1996

```
Boolean
  isDisabled() const;
```

Win	PM	Motif
Y	Y	Y

isFramed If the attribute framed (p. 594) is set, true is returned.

```
Boolean
  isFramed() const;
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support the framed attribute. This member function always returns false.

isHighlighted If the attribute highlighted (p. 594) is set, true is returned.

```
Boolean
  isHighlighted() const;
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support the highlighted attribute. This member function always returns false.

isNoDismiss If the attribute noDismiss (p. 594) is set, true is returned.

```
Boolean
  isNoDismiss() const;
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support the attribute noDismiss. This member function always returns false.

setChecked Displays a state indicator to the left of the menu item when the user selects it. The state indicator is removed when the user deselects the item.

```
IMenuItem&
  setChecked( Boolean checked = true );
```

Win	PM	Motif
Y	Y	Y

PM The state indicator is a check mark displayed to the left of the menu item.

Motif The state indicator is a square-shaped object displayed to the left of the menu item.

setDisabled Displays the menu item with unavailable-state emphasis, which means the user cannot select it.

```
IMenuItem&
  setDisabled( Boolean disabled = true );
```

Win	PM	Motif
Y	Y	Y

setFramed Draws a box to enclose the menu item.

```
IMenuItem&
  setFramed( Boolean framed = true );
```

Win	PM	Motif
I	*Y*	*I*

Motif Motif does not support the framed attribute.

setHighlighted

Displays the menu item with selected-state emphasis.

```
IMenuItem&
  setHighlighted( Boolean highlighted = true );
```

Win	PM	Motif
Y	*Y*	*I*

Motif Motif does not support the highlighted attribute.

setNoDismiss

Forces the menu item to remain visible when selected.

```
IMenuItem&
  setNoDismiss( Boolean noDismiss = true );
```

Win	PM	Motif
I	*Y*	*I*

Motif Motif does not support the noDismiss attribute.

Label

Use label members to set or query the label displayed on the menu item. The label can be either a bitmap or a text string.

bitmap

Returns the bitmap handle of the item.

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

Win	PM	Motif
Y	*Y*	*Y*

isBitmap

Returns true if this is a bitmap menu item.

```
Boolean
  isBitmap() const;
```

Win	PM	Motif
Y	*Y*	*I*

isText

Returns true if this is a menu item containing text.

```
Boolean
  isText() const;
```

Win	PM	Motif
Y	*Y*	*I*

setBitmap

Sets the bitmap to be displayed.

1

```
IMenuItem&
  setBitmap( const IResourceId& bitmapResId );
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

Use this version to load the bitmap from the resource library you specified on creation of the IResourceId (p. 808) object.

Use the following parameter:

bitmapResId
Resource identifier of the bitmap to display as a menu item.

2
```
IMenuItem&
  setBitmap( const IBitmapHandle& menuItem );
```

Win	PM	Motif
Y	Y	Y

Use this version to use a bitmap that you already have loaded. Use the following parameter:

menuItem Handle of the bitmap to display as a menu item.

3
```
IMenuItem&
  setBitmap( unsigned long bitmapResId );
```

Win	PM	Motif
Y	Y	Y

Use this version to load the bitmap from the default resource library.

Use the following parameter:

bitmapResId
Resource identifier of the bitmap to display as a menu item.

setText

Sets the text to be displayed. You can use either the resource identifier of the text or the actual text.

1
```
IMenuItem&
  setText( const char* newText );
```

Win	PM	Motif
Y	Y	Y

2
```
IMenuItem&
  setText( const IResourceId& textResId );
```

Win	PM	Motif
Y	Y	Y

text

Returns an IString (Vol. I) representing the text to be displayed.

```
IString
  text() const;
```

Win	PM	Motif
Y	Y	Y

Menu Layout

Use menu item settings to control how the menu is laid out around the menu item.

index Returns the ordinal position of the menu item in its submenu. The first item in a menu or submenu has an index value of 0.

```
unsigned long
  index() const;
```

Win	PM	Motif
Y	Y	Y

layoutType Returns the menu item placement and layout.

```
LayoutType
  layoutType() const;
```

Win	PM	Motif
Y	Y	Y

setIndex Sets the menu item ordinal position in its submenu. With this member function, you can add a menu item anywhere in the order.

```
IMenuItem&
  setIndex( unsigned long index = atEnd );
```

Win	PM	Motif
Y	Y	Y

setLayout Defines the menu item placement and layout. Use values from the enumeration IMenuItem::LayoutType (p. 597) to specify the layout type and placement.

```
IMenuItem&
  setLayout( LayoutType value );
```

Win	PM	Motif
Y	Y	Y

Properties

Use properties functions to obtain general information about the menu item.

id Returns the item identifier associated with the menu item.

```
unsigned long
  id() const;
```

Win	PM	Motif
Y	Y	Y

isDrawItem If the style drawItem (p. 595) is set, returns true.

```
Boolean
  isDrawItem() const;
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support the style drawItem. This member function always returns false.

isSelectable If the menu item is selectable, returns true.

© IBM Corp. 1992, 1996

```
Boolean
  isSelectable() const;
```

Win	PM	Motif
Y	Y	Y

isSeparator If the style separator (p. 596) is set, returns true.

```
Boolean
  isSeparator() const;
```

Win	PM	Motif
Y	Y	Y

setDrawItem Causes the system to generates a draw item event when the item needs drawing. This enables the application to draw the menu item.

```
IMenuItem&
  setDrawItem( Boolean drawItem = true );
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support the drawItem style.

setSelectable Enables the menu item for selection.

```
IMenuItem&
  setSelectable( Boolean selectable = true );
```

Win	PM	Motif
Y	Y	Y

setSeparator Makes the menu item a separator.

```
IMenuItem&
  setSeparator();
```

Win	PM	Motif
Y	Y	Y

Styles

For a menu item, *styles* represent properties of the menu item that typically do not change during the life of the item. *Attributes* represent properties that are more likely to be changed by the application during the life of the menu item.

IMenuItem defines a set of objects of the nested class IMenuItem::Style (p. 600). You can use these styles with constructors for IMenuItem, and IMenuItem::setDefaultStyle (p. 591).

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is IMenuItem::classDefaultStyle (p. 595) unless you have changed it using IMenuItem::setDefaultStyle (p. 591).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent menu items.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Submenu

Use submenu members to control the submenu behavior of a menu item. If a menu item is a submenu, a cascaded submenu is displayed when the user selects the menu item. You can use an IMenuHandler (p. 576) to alter the submenu before it is displayed.

isSubmenu Returns true if this item is a submenu.

```
Boolean
  isSubmenu() const;
```

Win	PM	Motif
Y	Y	I

setSubmenuHandle

Sets the menu item's submenu window handle. When the user selects the menu item, it displays the menu specified by *menuHandle* as a cascaded submenu.

```
IMenuItem&
  setSubmenuHandle( const IMenuHandle& menuHandle );
```

Win	PM	Motif
Y	Y	Y

submenuHandle

Returns the submenu window handle for the item. If the item does not have a submenu, 0 is returned.

```
IMenuHandle
  submenuHandle() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Implementation

These members provide utilities used to implement this class.

attribute Returns an unsigned long representing the menu item's attributes.

```
unsigned long
  attribute() const;
```

Win	PM	Motif
Y	Y	Y

extendedStyle

Returns an unsigned long value representing the menu item's extended style flags.

```
unsigned long
  extendedStyle() const;
```

Win	PM	Motif
Y	Y	I

helpId Returns the identifier of the help panel that is to be displayed when help is requested for the menu item. To display the help panel, you must first associate the window owning the menu which contains this menu item with a help instance. See the IHelpWindow (p. 442) class for more information.

```
unsigned long
  helpId() const;
```

Win	PM	Motif
Y	Y	Y

setAttribute Sets the menu item's attributes from the specified unsigned long.

```
IMenuItem&
  setAttribute( unsigned long newAttribute );
```

Win	PM	Motif
Y	Y	Y

setExtendedStyle

Sets the menu item's extended style to *newExtendedStyle*.

```
IMenuItem&
  setExtendedStyle( unsigned long newExtendedStyle );
```

Win	PM	Motif
Y	Y	I

setHelpId Stores the identifier of the help panel that is to be displayed when help is requested for the menu item. To display the help panel, you must first associate the window owning the menu which contains this menu item with a help instance. See the IHelpWindow (p. 442) class for more information.

```
IMenuItem&
  setHelpId( unsigned long newHelpTopicId );
```

Win	PM	Motif
Y	Y	Y

setStyle Sets the menu item's style from the specified unsigned long.

```
IMenuItem&
  setStyle( unsigned long newStyle );
```

Win	PM	Motif
Y	Y	Y

style Returns an unsigned long representing the menu item's style.

```
unsigned long
  style() const;
```

Win	PM	Motif
Y	Y	Y

Public Data

Attributes

For a menu item, *styles* represent properties of the menu item that typically do not change during the life of the item. *Attributes* represent properties that are more likely to be changed by the application during the life of the menu item. All attributes are set independently of each other.

IMenuItem defines a set of objects of the nested class IMenuItem::Attribute (p. 599). You can use these attributes with constructors for IMenuItem, and IMenuItem::setDefaultAttribute (p. 584).

checked Displays a state indicator to the left of the menu item when the user selects it. The state indicator is removed when the user deselects the item.

```
static const Attribute
  checked;
```

Win	PM	Motif
Y	Y	Y

PM The state indicator is a check mark displayed to the left of the menu item.

Motif The state indicator is a square-shaped object displayed to the left of the menu item.

© IBM Corp. 1992, 1996

classDefaultAttribute

Specifies the default attribute for this class, which is IMenuItem::noAttribute (p. 594).

```
static const Attribute
  classDefaultAttribute;
```

Win	PM	Motif
Y	*Y*	*Y*

disabled Displays the menu item with unavailable-state emphasis, which means the user cannot select it.

```
static const Attribute
  disabled;
```

Win	PM	Motif
Y	*Y*	*Y*

framed Encloses the menu item in a box.

```
static const Attribute
  framed;
```

Win	PM	Motif
I	*Y*	*I*

Motif This attribute is ignored in the Motif version.

highlighted Displays the menu item with selected-state emphasis.

```
static const Attribute
  highlighted;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This attribute is ignored in the Motif version.

noAttribute Specifies that no attribute applies to the menu item.

```
static const Attribute
  noAttribute;
```

Win	PM	Motif
Y	*Y*	*Y*

noDismiss Specifies that a submenu remains displayed when a user chooses a menu item in the submenu.

```
static const Attribute
  noDismiss;
```

Win	PM	Motif
I	*Y*	*I*

Motif This attribute is ignored in the Motif version.

Menu Layout

Use menu item settings to control how the menu is laid out around the menu item.

atEnd Use this constant to place the menu item at the end of the menu.

```
static const unsigned long
  atEnd;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

For a menu item, *styles* represent properties of the menu item that typically do not change during the life of the item. *Attributes* represent properties that are more likely to be changed by the application during the life of the menu item.

IMenuItem defines a set of objects of the nested class IMenuItem::Style (p. 600). You can use these styles with constructors for IMenuItem, and IMenuItem::setDefaultStyle (p. 591).

buttonSeparator

Draws a line to separate the menu item. The user cannot use cursor movement keys to get to it.

Note: You cannot use this style with the IMenuItem::split (p. 596) or IMenuItem::splitWithSeparator (p. 596) styles.

```
static const Style
  buttonSeparator;
```

Win	PM	Motif
I	*Y*	*I*

Motif This style is ignored in the Motif version.

classDefaultStyle

Specifies the original default style for this class, which is IMenuItem::noStyle (p. 595).

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

drawItem

Enables applications to draw the menu item.

```
static const Style
  drawItem;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This style is ignored in the Motif version.

noStyle

Specifies that no style applies to the menu item.

```
static const Style
  noStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

postHelp

Specifies that the menu item is a help menu.

© IBM Corp. 1992, 1996

```
static const Style
  postHelp;
```

Win	PM	Motif
I	*Y*	*Y*

PM When you use this style, the menu item generates a help message that is sent to the previous focus window. Therefore, the postHelp style works just like a **Help** push button.

Note: You cannot use this style with the IMenuItem::postSystemCommand (p. 596) style.

Motif When you use this style on a menu item that appears in the menu bar, the menu item appears to the far right of the menu bar.

postSystemCommand

Specifies that the menu item generate a system command message that can be handled by ICommandHandler::systemCommand (p. 217).

Note: You cannot use this style with the IMenuItem::postHelp (p. 595) style.

```
static const Style
  postSystemCommand;
```

Win	PM	Motif
I	*Y*	*Y*

separator

Specifies that the menu item is a separator.

```
static const Style
  separator;
```

Win	PM	Motif
Y	*Y*	*Y*

split

Starts the menu item in a new row or column.

Note: You cannot use this style with the IMenuItem::splitWithSeparator (p. 596) or IMenuItem::buttonSeparator (p. 595) styles.

```
static const Style
  split;
```

Win	PM	Motif
Y	*Y*	*Y*

splitWithSeparator

Starts the menu item in a new row or column with a line drawn to separate it.

Note: You cannot use this style with the IMenuItem::split (p. 596) or IMenuItem::buttonSeparator (p. 595) styles.

```
static const Style
  splitWithSeparator;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This style is ignored in the Motif version.

unavailable Sets unavailable-state emphasis for a menu item.

```
static const Style
  unavailable;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IMenuItem contains the following nested classes:

IMenuItem::Style (see page 600)
IMenuItem::Attribute (see page 599)

CommandType

```
CommandType {
    command,
    systemCommand,
    helpCommand
    };
```

These enumerators specify the type of event generated when a user selects the menu item:

command
Generates an ICommandHandler::command (p. 217) event.

systemCommand
Generates an ICommandHandler::systemCommand (p. 217) event.

helpCommand
Generates a help message.

LayoutType

```
LayoutType {
    normalLayout,
    splitLayout,
    splitWithSeparatorLayout,
    buttonSeparatorLayout
    };
```

These enumerators specify the placement and layout of the menu item:

© IBM Corp. 1992, 1996

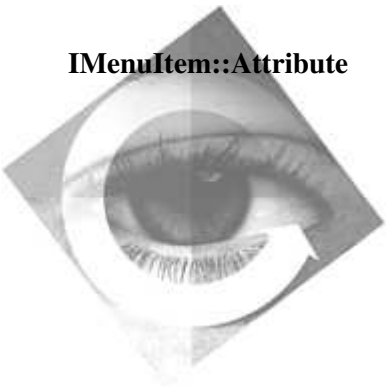
normalLayout
Specifies that no style applies.

splitLayout
Starts the menu item in a new row or column.

splitWithSeparatorLayout
Starts the menu item in a new row or column with a line drawn to separate it.

buttonSeparatorLayout
Separates the menu item and does not allow cursor movement to it.

Motif Motif does not support the enumerators splitWithSeparatorLayout and buttonSeparatorLayout.



IMenuItem::Attribute

Derivation

IBase
  IBitFlag
    IMenuItem::Attribute

Inherited By None.

Header File imnitem.hpp

The nested class IMenuItem::Attribute provides a set of valid menu item attributes for the class IMenuItem (p. 582).

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

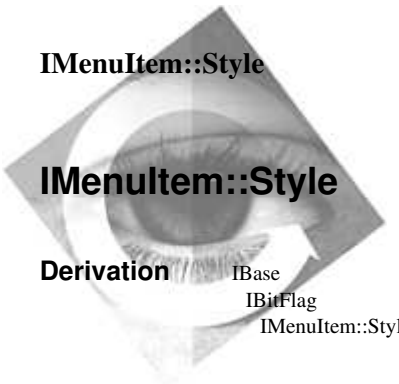
Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



IMenuItem::Style

Derivation

IBase
IBitFlag
IMenuItem::Style

Inherited By None.

Header File imnitem.hpp

The nested class IMenuItem::Style represents properties of a menu item. The class IMenuItem (p. 582) defines IMenuItem::Style objects (p. 593) that you can use when constructing an IMenuItem object.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IMenuNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        IMenuNotifyHandler

Inherited By None.

Header File imenunhd.hpp

Members

Member	Page
Constructor	601
dispatchHandlerEvent	602
~IMenuNotifyHandler	602

IMenuNotifyHandler objects process events for all classes of menus.

This class is designed to handle events that require the menu class to generate a notification. If notifications are enabled for the menu object, a notification will be generated and sent to all its observers when the proper conditions for the specific notification exist.

If you create a class derived from IMenu (p. 525) that needs to notify observers of additional events, create a class derived from IMenuNotifyHandler that overrides dispatchHandlerEvent (p. 602). In the class derived from IMenu, implement enableNotification (p. 540) to call IWindow::setNotificationHandler (p. 1092), passing it an object of the class derived from IMenuNotifyHandler.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMenuNotifyHandler

Default constructor.

© Copyright IBM Corp. 1992, 1996

IMenuNotifyHandler

Note: Generally you do not need to construct an object of this class. Calling IMenu::enableNotification (p. 540) causes an IMenuNotifyHandler object to be constructed and added to the window, if necessary.

```
IMenuNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IMenuNotifyHandler

```
virtual
  ~IMenuNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

Notification handlers process events that are sent or posted to a window by calling observer objects interested in those events.

dispatchHandlerEvent

Notifies the menu observers when it receives any of the following events:

- Foreground color event
- Background color event
- Disabled foreground color event

- Disabled background color event
- Highlight foreground color event
- Highlight background color event
- System command event

This function also calls IWindowNotifyHandler::dispatchHandlerEvent (p. 1112) so that observers will also be notified of generic window events.

If you create a class derived from IMenuNotifyHandler, its dispatchHandlerEvent function should call IMenuNotifyHandler::dispatchHandlerEvent for events it does not process.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IMessageBox

Derivation

IBase
IVBase
IMessageBox

Inherited By None.

Header File imsgbox.hpp

Members

Member	Page	Member	Page
Constructor	604	noIcon	610
abortRetryIgnoreButton	608	okButton	610
applicationModal	608	okCancelButton	610
cancelButton	608	queryIcon	610
defButton1	608	retryCancelButton	610
defButton2	609	setTitle	607
defButton3	609	show	605
enterButton	609	systemModal	610
enterCancelButton	609	warningIcon	610
errorIcon	609	yesNoButton	611
informationIcon	609	yesNoCancelButton	611
moveable	609	~IMessageBox	605

The IMessageBox class displays message boxes that are compliant with both the Common User Access (CUA) and the Motif style guides.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IMessageBox objects because both the copy constructor and assignment operator are private functions.

IMessageBox

```
IMessageBox( IWindow* owner );
```

Win	PM	Motif
Y	Y	Y

You can only construct objects of this class by specifying an object of the class IWindow (p. 1044).

© Copyright IBM Corp. 1992, 1996

owner The IWindow object that becomes the owner of the new message box. The message box title defaults to the owner's title. You can specify that the message box has no owner window by using a value of 0.

To display help for a message box, ensure that *owner* is nonzero, and that the owner specified is associated with an IHelpWindow (p. 442) object.

~IMessageBox

```
virtual
  ~IMessageBox();
```

Win	PM	Motif
Y	Y	Y

Showing the Message Box

A message window is not displayed when you construct an IMessageBox object. When you show the message window, you can specify its message text, push buttons, icon, and other specifics.

show

Shows a message box. The returned value is an enumerator provided by Response (p. 611).

1

```
virtual IMessageBox::Response
  show( const IResourceId& message,
        const Style& style,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

message Identifier for a string or message resource to be used for the message box text.

style The style of the message box, as a combination of objects of the class IMessageBox::Style (p. 608).

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

2

```
virtual IMessageBox::Response
  show( const char* message,
        Severity severity,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

message The message box text.

© IBM Corp. 1992, 1996

severity Use the enumeration Severity (p. 612) to specify the severity of your message.

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

3
```
virtual IMessageBox::Response
  show( const IResourceId& message,
        Severity severity,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

message Identifier for a string or message resource to be used for the message box text.

severity Use the enumeration Severity (p. 612) to specify the severity of your message.

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

4
```
virtual IMessageBox::Response
  show( const char* message,
        const Style& style,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

message The message box text.

style The style of the message box, as a combination of objects of the class IMessageBox::Style (p. 608).

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

5
```
virtual IMessageBox::Response
  show( const IException& exception,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

exception An IException (Vol. I) object. The message box displays the text contained in this object.

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

6
```
virtual IMessageBox::Response
  show( const IBaseErrorInfo& error,
        unsigned long helpId = 0 );
```

Win	PM	Motif
Y	Y	Y

error An IBaseErrorInfo (Vol. I) object. The message box displays the text contained in this object.

helpPanelId
If *helpPanelId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpPanelId* parameter is displayed. Optional.

W32s The helpPanelId is ignored and no help button is shown.

Title Text

Use these functions to set the text displayed in the title bar of the message box, or if the message box has no title bar, to set the text displayed above the message text.

setTitle

Sets the message box's title.

1
```
virtual IMessageBox&
  setTitle( const IResourceId& title );
```

Win	PM	Motif
Y	Y	Y

title The new title. The text is the specified string resource in a string table.

2
```
virtual IMessageBox&
  setTitle( const char* title );
```

Win	PM	Motif
Y	Y	Y

title The new title. The text is the specified character string.

© IBM Corp. 1992, 1996

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Public Data

Styles

IMessageBox defines a set of objects of the nested class IMessageBox::Style (p. 613). You can use these styles with versions of the function show (p. 605) to identify the push buttons and icon to be displayed on the message box, the modality of the message box, and whether the user is allowed to move the message box.

abortRetryIgnoreButton

Specifies the message box should contain the **Abort**, **Retry**, and **Ignore** push buttons.

```
static const Style
  abortRetryIgnoreButton;
```

Win	PM	Motif
Y	*Y*	*Y*

applicationModal

Specifies that the message box be application-modal. This is the default style.

Note: The owner window of an application-modal message box is disabled while the message box is displayed.

```
static const Style
  applicationModal;
```

Win	PM	Motif
Y	*Y*	*Y*

cancelButton

Specifies the message box should contain a **Cancel** push button.

```
static const Style
  cancelButton;
```

Win	PM	Motif
I	*Y*	*Y*

defButton1

Specifies that the default selection for the message box should be the first push button. This is the default style.

```
static const Style
  defButton1;
```

Win	PM	Motif
Y	*Y*	*Y*

defButton2 Specifies that the default selection for the message box should be the second push button.

```
static const Style
  defButton2;
```

Win	PM	Motif
Y	*Y*	*Y*

defButton3 Specifies that the default selection for the message box should be the third push button.

```
static const Style
  defButton3;
```

Win	PM	Motif
Y	*Y*	*Y*

enterButton Specifies the message box should contain an **Enter** push button.

```
static const Style
  enterButton;
```

Win	PM	Motif
I	*Y*	*Y*

enterCancelButton

Specifies the message box should contain both **Enter** and **Cancel** push buttons.

```
static const Style
  enterCancelButton;
```

Win	PM	Motif
I	*Y*	*Y*

errorIcon Specifies the message box should contain a icon.

```
static const Style
  errorIcon;
```

Win	PM	Motif
Y	*Y*	*Y*

informationIcon

Specifies the message box should contain an icon.

```
static const Style
  informationIcon;
```

Win	PM	Motif
Y	*Y*	*Y*

moveable Specifies that the user can move the message box. This style gives the message box a title bar.

© IBM Corp. 1992, 1996

```
static const Style
  moveable;
```

Win	PM	Motif
Y	Y	Y

noIcon Specifies the message box should contain no icon.

```
static const Style
  noIcon;
```

Win	PM	Motif
Y	Y	Y

okButton Specifies the message box should contain an **OK** push button.

```
static const Style
  okButton;
```

Win	PM	Motif
Y	Y	Y

okCancelButton

Specifies the message box should contain both **OK** and **Cancel** push buttons.

```
static const Style
  okCancelButton;
```

Win	PM	Motif
Y	Y	Y

queryIcon Specifies the message box should contain a icon.

```
static const Style
  queryIcon;
```

Win	PM	Motif
Y	Y	Y

retryCancelButton

Specifies the message box should contain both **Retry** and **Cancel** push buttons.

```
static const Style
  retryCancelButton;
```

Win	PM	Motif
Y	Y	Y

systemModal Specifies that the message box be system modal. This prevents the user from interacting with any other window, including those of other applications, until dismissing the message box.

```
static const Style
  systemModal;
```

Win	PM	Motif
I	Y	Y

warningIcon Specifies the message box should contain an icon.

```
static const Style
  warningIcon;
```

Win	PM	Motif
Y	Y	Y

yesNoButton Specifies the message box should contain both **Yes** and **No** push buttons.

```
static const Style
  yesNoButton;
```

Win	PM	Motif
Y	Y	Y

yesNoCancelButton

Specifies the message box should contain **Yes**, **No**, and **Cancel** push buttons.

```
static const Style
  yesNoCancelButton;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IMessageBox contains the following nested classes:

IMessageBox::Style (see page 613)

Response

```
Response {
    enter,   ok,      cancel,  abort,   retry,
    ignore,  yes,     no,      unknown
    };
```

These enumerators identify the push button used by the user to dismiss a message box.

enter
The user selected the **Enter** push button.

ok
The user selected the **OK** push button.

cancel
The user selected the **Cancel** push button or closed the message box using its system menu.

© IBM Corp. 1992, 1996

abort
The user selected the **Abort** push button.

retry
The user selected the **Retry** push button.

ignore
The user selected the **Ignore** push button.

yes
The user selected the **Yes** push button.

no
The user selected the **No** push button.

unknown
The user selected a push button other than one of the above.

Severity

```
Severity {
    information,  warning,      action,       critical,     catastrophic
    };
```

Use these enumerators to specify the different severity levels of a message box:

information
Displays an information icon and an **OK** push button.

warning
Displays a warning icon, and **Enter** and **Cancel** push buttons.

action
Displays a query icon, and **Retry** and **Cancel** push buttons.

critical
Displays an error icon, and **Retry** and **Cancel** push buttons.

catastrophic
Displays an error icon and an **OK** push button.

IMessageBox::Style

Derivation

IBase
  IBitFlag
    IMessageBox::Style

Inherited By None.

Header File imsgbox.hpp

The IMessageBox::Style class represents characteristics of a message box. The class IMessageBox (p. 604) defines IMessageBox::Style objects (p. 608) to identify the push buttons and icon to be displayed on the message box, the modality of the message box, and whether the user is allowed to move the message box.

Unlike the style objects defined by classes derived from IWindow, you do not specify IMessageBox::Style objects when constructing an IMessageBox object. Instead you can use these styles with versions of the function IMessageBox::show (p. 605).

Motif The User Interface Class Library supplies the message catalog, ibmcl.msg. This catalog contains the button text for the strings Abort, Retry, Ignore and Enter, which is used with the IMessageBox::enterCancelButton (p. 609), IMessageBox::retryCancelButton (p. 610), and IMessageBox::abortRetryIgnoreButton (p. 608) styles. If you use these styles, the User Interface Class Library attempts to load the text strings for the buttons from the message catalog. If the message catalog entry for the button text cannot be found, the library uses the English words Abort, Retry, Ignore, and Enter in the **Abort**, **Retry**, **Ignore**, and **Enter** push buttons, respectively.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© Copyright IBM Corp. 1992, 1996

IMessageBox::Style

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IMessageQueueHandle

Derivation

IBase
  IHandle
    IMessageQueueHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page	Member	Page
Constructor	615	postEvent	616
operator Value	616	postEvents	616

The IMessageQueueHandle class accesses application message queues.

PM IMessageQueueHandle is an alias for the OS/2 Programmer's Toolkit HMQ typedef.

Win The system creates the message queue automatically for a thread. The message queue handle is the thread ID of the thread.

Motif AIX does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IMessageQueueHandle

Constructs objects of this class from a message queue handle (a value of type IHandle::Value), which defaults to 0.

```
IMessageQueueHandle( Value hmq = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

© Copyright IBM Corp. 1992, 1996

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
N	N	Y

Post Event

Use these members to post an event to message queue handles.

postEvent Posts an event constructed from the parameters to the message queue handle object or other message queue handles.

1
```
static void
  postEvent( const IMessageQueueHandle& handle,
             unsigned long eventId,
             const IEventParameter1& parm1 = 0ul,
             const IEventParameter2& parm2 = 0ul );
```

Win	PM	Motif
Y	Y	Y

2
```
void
  postEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0ul,
             const IEventParameter2& parm2 = 0ul ) const;
```

Win	PM	Motif
Y	Y	Y

postEvents Broadcasts an event to other message queue handles asynchronously.

```
static void
  postEvents( unsigned long eventId,
              const IEventParameter1& parm1 = 0ul,
              const IEventParameter2& parm2 = 0ul );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

IModuleHandle

Derivation

IBase
IHandle
IModuleHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	618
operator Value	618

The IModuleHandle class accesses application modules.

PM IModuleHandle is an alias for the OS/2 Programmer's Toolkit HMODULE typedef.

Motif AIX does not support this class.

Public Functions

Constructors

You can construct objects of this class.

IModuleHandle

Constructs objects of this class from a module handle (a value of type IHandle::Value), which defaults to 0.

```
IModuleHandle( Value hmod = 0 );
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

© Copyright IBM Corp. 1992, 1996

```
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**	

© IBM Corp. 1992, 1996

IMouseClickEvent

Derivation

IBase
IVBase
IEvent
IMouseEvent
IMouseClickEvent

Inherited By None.

Header File imousevt.hpp

Members

Member	Page	Member	Page
Constructor	620	mouseButton	621
mouseAction	621	~IMouseClickEvent	621

The IMouseClickEvent class represents mouse click events. Objects of this class are constructed by a mouse handler when the user presses, releases, clicks, or double-clicks a mouse button.

Mouse click events are first dispatched to the window that the mouse pointer is positioned on. If that window does not process the mouse event, the event is dispatched to its owner window. The event continues to be dispatched to the next owner window until a handler stops the processing or a window processes the mouse event itself.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMouseClickEvent

Constructs an IMouseClickEvent object from the specified event. IMouseHandler::dispatchHandlerEvent (p. 633) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function IMouseHandler::mouseClicked (p. 633).

```
IMouseClickEvent( const IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

© Copyright IBM Corp. 1992, 1996

~IMouseClickEvent

```
virtual
  ~IMouseClickEvent();
```

Win	PM	Motif
Y	Y	Y

Mouse Action

A mouse button is either in an up state or down state. This class allows you to determine the action used to change the state of a mouse button, whether the user pressed a mouse button, released a mouse button, clicked a mouse button (pressed and released within a predefined time), or double-clicked a mouse button.

mouseAction

Returns the action taken on the mouse.

```
MouseAction
  mouseAction() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The event does not contain a recognized action. The sender of the event may have constructed it incorrectly.

Mouse Button

IMouseClickEvent identifies the mouse button that the user pressed or released. This allows you to provide different actions for different mouse buttons.

mouseButton

Returns the mouse button that generated the event.

```
MouseButton
  mouseButton() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The event does not identify a mouse button. The sender of the event may have constructed it incorrectly.

Inherited Public Functions

IMouseEvent		
isAltKeyDown	isCtrlKeyDown	isShiftKeyDown

© IBM Corp. 1992, 1996

IMouseClickEvent

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**	

MouseAction

```
MouseAction {
    click,
    doubleClick,
    down,
    up
    };
```

The mouse action enumerations are:

click	A mouse button was pressed and released.
doubleClick	A mouse button was pressed and released twice in succession.
down	A mouse button was pressed.
up	A mouse button was released.

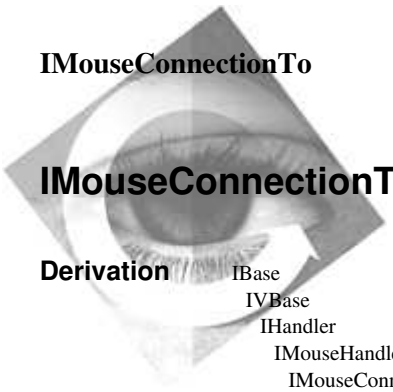
MouseButton

```
MouseButton {
    button1,              button2,              button3,
    buttonChord,          button12 = buttonChord
    };
```

The mouse button enumerations are as follows:

button1	Mouse button 1.
button2	Mouse button 2.
button3	Mouse button 3.
buttonChord	Mouse buttons 1 and 2 are pressed together.

© IBM Corp. 1992, 1996



IMouseConnectionTo

Derivation

IBase
IVBase
IHandler
IMouseHandler
IMouseConnectionTo

Inherited By None.

Header File imoushdr.hpp

Members

Member	Page	Member	Page
Constructor	625	mousePointerChange	626
mouseClicked	626	~IMouseConnectionTo	625
mouseMoved	626		

The IMouseConnectionTo class is a template class, derived from IMouseHandler (p. 631), that processes mouse events. This class allows you to process mouseMoved, mouseClicked, and mousePointerChange events in objects that do not derive from IMouseHandler without having to manually derive from IMouseHandler, override the mouseMoved, mouseClicked, or mousePointerChange functions, and pass the events to the object.

To use the IMouseConnectionTo class to process mouseMoved, mouseClicked, and mousePointerChange events, follow these steps:

1. Instantiate the IMouseConnectionTo template with a class containing member functions whose signature and behavior are the same as IMouseHandler::mouseMoved (p. 633), IMouseHandler::mouseClicked (p. 633), and IMouseHandler::mousePointerChange (p. 633),
2. Construct an object of the new template class by passing the object and the address of all three functions handling the corresponding mouse events.
3. Attach the template mouse handler by using IHandler::handleEventsFor (p. 413) to pass the appropriate window to the mouse handler.

You can also process one or two out of the three mouse events by specifying a null value on the constructor for the address of the members that you do not want to process.

© Copyright IBM Corp. 1992, 1996

You use an IMouseConnectionTo object anywhere you use an IMouseHandler object. See the class description of IMouseHandler (p. 631) for a description of the uses and limitations of these classes.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign objects of this class.

IMouseConnectionTo

Constructs the IMouseConnectionTo object.

```
IMouseConnectionTo(
   ATarget& target,
   ClickMemberFunction clickMemberFunction = 0,
   MoveMemberFunction moveMemberFunction = 0,
   PointerMemberFunction pointerMemberFunction = 0 );
```

Win	PM	Motif
Y	*N*	*N*

The parameters are the following:

target The object to receive mouse events.

clickMemberFunction
The address of the member function to receive mouse click events or 0 if you do not want to process mouse click events.

moveMemberFunction
The address of the member function to receive mouse move events or 0 if you do not want to process mouse move events.

pointerMemberFunction
The address of the member function to receive mouse pointer changed events or 0 if you do not want to process mouse pointer changed events.

~IMouseConnectionTo

```
virtual
  ~IMouseConnectionTo();
```

Win	PM	Motif
Y	*N*	*N*

© IBM Corp. 1992, 1996

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Processing

These members are overridden to route mouse events to the target object that you provided on construction.

mouseClicked

Overridden to route mouse clicked events to the *clickMemberFunction* of the target object if you provided one when constructing the handler. Otherwise, it returns false.

```
virtual Boolean
  mouseClicked( IMouseClickEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

mouseMoved

Overridden to route mouse moved events to the *moveMemberFunction* of the target object if you provided one when constructing the handler. Otherwise, it returns false.

```
virtual Boolean
  mouseMoved( IMouseEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

mousePointerChange

Overridden to route mouse pointer changed events to the *pointerMemberFunction* of the target object if you provided one when constructing the handler. Otherwise, it returns false.

```
virtual Boolean
  mousePointerChange( IMousePointerEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

Inherited Protected Functions

IMouseHandler		
dispatchHandlerEvent	mouseClicked	mouseMoved

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

PointerMemberFunction

```
typedef Boolean (ATarget::*PointerMemberFunction)(IMousePointerEvent&);
```

A pointer to a member function of Class ATarget that takes a reference to an IMousePointerEvent and returns a Boolean.

MoveMemberFunction

```
typedef Boolean (ATarget::*MoveMemberFunction)(IMouseEvent&);
```

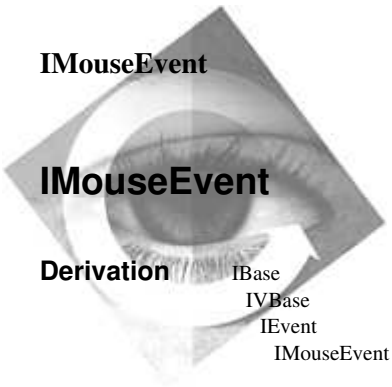
A pointer to a member function of Class ATarget that takes a reference to an IMouseEvent and returns a Boolean.

ClickMemberFunction

```
typedef Boolean (ATarget::*ClickMemberFunction)(IMouseClickEvent&);
```

A pointer to a member function of Class ATarget that takes a reference to an IMouseClickEvent and returns a Boolean.

© IBM Corp. 1992, 1996



IMouseEvent

Derivation

IBase
IVBase
IEvent
IMouseEvent

Inherited By IMouseClickEvent

Header File imousevt.hpp

Members

Member	Page	Member	Page
Constructor	628	mousePosition	629
isAltKeyDown	629	windowUnderPointer	630
isCtrlKeyDown	629	~IMouseEvent	629
isShiftKeyDown	629		

The IMouseEvent class represents a mouse move event and is created when the user moves the mouse. You can use this event to query the location of the mouse and the window under the mouse at the time the mouse move event occurred. You can also query the keyboard state of the Alt key (on some keyboards, this is called the Menu key), the Ctrl (Control) key, or the Shift key at the time the mouse move event occurred.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMouseEvent Constructs an IMouseEvent object from the specified event. IMouseHandler::dispatchHandlerEvent (p. 633) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function IMouseHandler::mouseMoved (p. 633).

```
IMouseEvent( const IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

© Copyright IBM Corp. 1992, 1996

~IMouseEvent

```
virtual
  ~IMouseEvent();
```

Win	PM	Motif
Y	Y	Y

Keyboard State

You can query the keyboard state information at the time the mouse event was generated. This allows you to detect certain combinations of the keyboard and mouse, such as when the user presses mouse button 1 with the Alt key pressed.

isAltKeyDown

Returns true if the Alt key is down when the mouse event was generated. On some keyboards, this Alt key is called the Menu key.

```
Boolean
  isAltKeyDown() const;
```

Win	PM	Motif
Y	Y	Y

isCtrlKeyDown

Returns true if the Ctrl key is down when the mouse event was generated.

```
Boolean
  isCtrlKeyDown() const;
```

Win	PM	Motif
Y	Y	Y

isShiftKeyDown

Returns true if the Shift key is down when the mouse event was generated.

```
Boolean
  isShiftKeyDown() const;
```

Win	PM	Motif
Y	Y	Y

Position

These members query the mouse's position. The position information is useful in determining what the user is interacting with.

mousePosition

Returns the mouse pointer position relative to the window returned by IEvent::window (p. 312). If you need to obtain the mouse position relative to the desktop window, use the static function IWindow::mapPoint (p. 1078).

© IBM Corp. 1992, 1996

IMouseEvent

```
IPoint
  mousePosition() const;
```

Win	PM	Motif
Y	Y	Y

Window Information

Use these members to determine the window that the mouse is interacting with.

windowUnderPointer

Returns the handle of the window that is under the mouse pointer.

```
IWindowHandle
  windowUnderPointer() const;
```

Win	PM	Motif
Y	Y	Y

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**	

IMouseHandler

Derivation

IBase
  IVBase
    IHandler
      IMouseHandler

Inherited By

IMouseConnectionTo
IMousePointerHandler

Header File

imoushdr.hpp

Members

Member	Page	Member	Page
Constructor	632	mouseMoved	633
dispatchHandlerEvent	633	mousePointerChange	633
mouseClicked	633	~IMouseHandler	632

The IMouseHandler class processes a variety of mouse events. These events include button presses and releases, double-clicks, multiple button presses, and mouse movement.

You can attach an IMouseHandler object to any kind of window. Although the control or window that the mouse is over is the first to receive a mouse event, any event that is passed on for additional processing will likely be dispatched to the owner window.

A mouse event continues to travel up the owner window chain until either a handler stops it or the event is processed by the window itself. As a result, an IMouseHandler object should not stop the processing of any mouse event that it does not need to handle because the event has the potential of being handled by any window in the owner chain, such as the frame window.

When an IMouseHandler object receives a mouse event, the IMouseHandler object creates either an IMouseEvent (p. 628), an IMouseClickEvent (p. 620), or an IMousePointerEvent (p. 635) object and then routes it to a virtual function of the mouse handler. Override these virtual functions to supply your own processing of a mouse event.

The return value from the virtual function is interpreted as follows:

© Copyright IBM Corp. 1992, 1996

true	The mouse event requires no additional processing and is not passed to another handler.
false	The mouse event is passed on for additional processing when the following situations occur: • If there is another handler for the window, the event is passed on to the next handler. • If this is the last handler for the window, the event is passed on a call to the window's defaultProcedure (p. 1082) function. This could result in the event being dispatched to the window's owner window, where the processing of the mouse event starts again.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMouseHandler

Default constructor.

```
IMouseHandler();
```

Win	PM	Motif
Y	Y	Y

~IMouseHandler

```
virtual
  ~IMouseHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IMouseHandler object processes only mouse-related events.

dispatchHandlerEvent

If a mouse-related event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

A mouse handler contains event-processing members that you use to process mouse-related events. You should override at least one of these virtual functions in a derived class.

mouseClicked

Implemented by derived classes to process a mouse-click event.

```
virtual Boolean
  mouseClicked( IMouseClickEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

mouseMoved Implemented by derived classes to process a mouse-movement event.

```
virtual Boolean
  mouseMoved( IMouseEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

mousePointerChange

Implemented by derived classes to change the appearance of the mouse pointer when it is over the handled window. IMouseHandler::dispatchHandlerEvent (p. 633) calls this function when the mouse pointer is moved over the handled window. You do not need to override this function if you change the mouse pointer for a frame

© IBM Corp. 1992, 1996

window and all its children. Instead, use the function IFrameWindow::setMousePointer (p. 367).

```
virtual Boolean
  mousePointerChange( IMousePointerEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IMousePointerEvent

Derivation

IBase
  IVBase
    IEvent
      IMousePointerEvent

Inherited By None.

Header File imousevt.hpp

Members

Member	Page	Member	Page
Constructor	635	windowId	636
defaultMousePointer	636	~IMousePointerEvent	636
setMousePointer	636		

The IMousePointerEvent class represents the event when the pointer moves over a window. This event allows you to query the pointer that the system will show when it is over a window and allows you to change the pointer to show when it is over a window.

Mouse pointer events are first dispatched to the window that the mouse pointer is positioned on. If that window does not process the mouse event, the event is dispatched to its owner window. The event continues to be dispatched to the next owner window until a handler stops the processing or a window processes the mouse pointer event itself.

PM An IMousePointerEvent is created from a WM_CONTROLPOINTER message.

Win An IMousePointerEvent is created from a WM_SETCURSOR message.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMousePointerEvent

Constructs an IMousePointerEvent object from the specified event. IMouseHandler::dispatchHandlerEvent (p. 633) constructs objects of this class from

© Copyright IBM Corp. 1992, 1996

IMousePointerEvent

an object of the class IEvent (p. 304) and passes the resulting object to the function IMouseHandler::mousePointerChange (p. 633).

```
IMousePointerEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	N

~IMousePointerEvent

```
virtual
  ~IMousePointerEvent();
```

Win	PM	Motif
Y	Y	N

Mouse Pointer

The mouse pointer can have different appearances, such as an arrow or a clock, to indicate the kind of user interaction a window allows. Use these members to set and query the mouse pointer that the window under the mouse will use.

defaultMousePointer

Returns the mouse pointer that the system would show for the window that is under the mouse if you did not call setMousePointer (p. 636).

```
IPointerHandle
  defaultMousePointer() const;
```

Win	PM	Motif
Y	Y	N

setMousePointer

Sets the pointer to use for the window that is under the mouse.

```
IMousePointerEvent&
  setMousePointer( const IPointerHandle& mousePointer );
```

Win	PM	Motif
Y	Y	N

Window Information

Use these members to determine the window that is under the mouse.

windowId

Returns the window identifier of the window that is under the mouse.

```
unsigned long
  windowId() const;
```

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

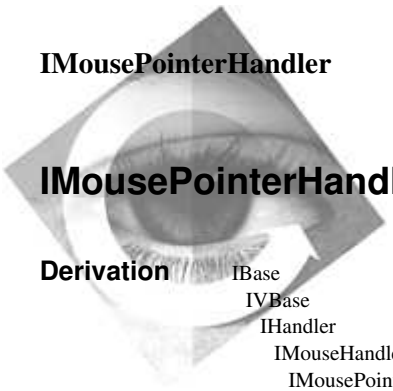
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IMousePointerHandler

Derivation

IBase
IVBase
IHandler
IMouseHandler
IMousePointerHandler

Inherited By None.

Header File imphdr.hpp

Members

Member	Page	Member	Page
Constructor	639	mousePointerChange	640
defaultHandler	639	~IMousePointerHandler	639

The IMousePointerHandler class sends mouse pointer events to the owner window of the window that the handler is attached to.

By default, the system sends mouse pointer events to the owner of the window that the mouse is over. Typically, the owner window allows the default procedure of the window to handle mouse pointer events. The default behavior is to leave the mouse pointer unchanged.

It is often convenient to change the mouse pointer on a frame window-by-frame window basis so that child windows that exist in a frame window use the pointer specified for the frame window. You can change the mouse pointer for a frame window and its children by using IFrameWindow::setMousePointer (p. 367). To accomplish this behavior, all mouse pointer events that are generated for a frame window and its children must eventually be sent to the frame window. By attaching this handler to the children of a frame window, the mouse pointer events are sent up the owner chain where they can be handled by the frame window.

The User Interface Class Library automatically attaches this handler to controls that require sending the mouse pointer event to its owning window. Use this handler if you intend to call IFrameWindow::setMousePointer and you are creating a custom control that is a child of the frame window for which you want to change the pointer.

PM An IMousePointerEvent is created from a WM_CONTROLPOINTER message.

Win An IMousePointerEvent is created from a WM_SETCURSOR message.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

You can construct and destruct objects of this class.

IMousePointerHandler

Provides the default constructor.

There is no need for you to create an object of this class, however. Instead, use the static function IMousePointerHandler::defaultHandler (p. 639) to obtain a pointer to a static object of this class.

```
IMousePointerHandler();
```

Win	PM	Motif
Y	*Y*	*N*

~IMousePointerHandler

```
virtual
  ~IMousePointerHandler();
```

Win	PM	Motif
Y	*Y*	*N*

Static Object

You can use this class by accessing a static object of this class. You can attach this object to the appropriate windows in your application.

defaultHandler

Returns a pointer to the static object of this class.

```
static IMousePointerHandler*
  defaultHandler();
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

© IBM Corp. 1992, 1996

IMousePointerHandler

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Processing

A mouse-pointer handler provides a specific implementation for a mouse handler by overriding IMouseHandler (p. 631) virtual functions.

mousePointerChange

This function processes a pointer-change event by sending it to the event window's owner window.

```
virtual Boolean
  mousePointerChange( IMousePointerEvent& event );
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Protected Functions

IMouseHandler		
dispatchHandlerEvent	mouseClicked	mouseMoved

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IMultiLineEdit

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IMultiLineEdit

Inherited By

None.

Header File

imle.hpp

Members

Member	Page	Member	Page
Constructor	647	hasTextChanged	643
add	658	horizontalScroll	665
addAsLast	658	ignoreTab	665
addAtOffset	659	importFromFile	649
addLine	650	isDragStarting	662
addLineAsLast	650	isUndoable	661
backgroundColor	646	isWordWrap	661
border	664	isWriteable	653
border3D	665	limit	659
classDefaultStyle	665	limitId	664
clear	644	numberOfLines	651
convertToGUIStyle	657	passEventToOwner	663
copy	644	paste	646
cursorLinePosition	651	readOnly	665
cursorPosition	651	registerCallbacks	663
cut	645	removeAll	660
dataUpdateId	664	removeLine	651
defaultStyle	657	resetTextChangedFlag	643
deselect	664	selectedRange	654
disableDataUpdate	653	selectedText	654
disableUpdate	653	selectedTextLength	654
disableWordWrap	661	selectRange	655
discard	645	setCursorLinePosition	651
editRegionHeight	655	setCursorPosition	652
editRegionWidth	655	setDefaultStyle	657
enableDataUpdate	653	setEditRegion	655
enableNotification	652	setEditRegionHeight	656
enableUpdate	653	setEditRegionWidth	656
enableWordWrap	661	setFont	648
end	664	setLimit	659
exportSelectedTextToFile	649	setTab	657
exportToFile	649	setText	659
foregroundColor	647	setTextChangedFlag	644
hasSelectedText	654	setTop	652

© Copyright IBM Corp. 1992, 1996

IMultiLineEdit

Member	Page	Member	Page
text	660	unregisterCallbacks	663
textLength	660	verticalScroll	665
top	652	visibleLines	652
topHandle	647	wordWrap	666
undo	661	~IMultiLineEdit	648

The IMultiLineEdit class creates and manages multiline edit (MLE) controls. This control is similar to an entry field, but it lets the user enter multiple lines of text rather than just one.

You derive classes from the following handlers and attach them to an MLE control:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- IScrollHandler (p. 845)

The MLE control can enable screen refreshes during its member functions processing. In doing so, it negates any previous use of IMultiLineEdit::disableUpdate.

Note: Do not load text containing nulls or any other embedded nonprintable characters into an IMultiLineEdit object. Because the user cannot see these characters, the user is likely to type over or delete them.

PM When exporting a file with a format of cfText, a CR-LF sequence is counted as one character. However, importFromFile reads the file in binary so that the CR-LF is counted as two bytes. This makes the size of the imported files larger than the exported file. This can create a problem for very small MLEs. An exception is thrown indicating that the string being inserted into the MLE is larger than the set limit.

You should use the noTran format of the EOLFormat or derive customized import or export functions to handle this condition.

Motif The User Interface Class Library implements this class with the XmText widget. The XmText widget does not support embedded NULLs or embedded nonprintable characters. Only the text preceding the first NULL is loaded into the control.

AIX only supports the noTran format enumerator. If you specify any other EOLFormat (p. 666), the member functions ignore it and use noTran instead.

The Motif XmText widget does not support line-oriented functions; the User Interface Class Library has added this support. However, it does not make fine adjustments for a proportional font (to ensure good performance). This means that if a proportional font is used in the control, the number of a line on the screen might not match the calculated line number used internally by the line-oriented functions. Therefore, if you use a proportional font in the control, use character, position-oriented functions instead.

Line-oriented functions may not work if tab characters are used. Typically, Motif converts tabs as 10 blank characters, but if a line ends with a tab, it might cause an incorrect line count.

Handlers derived from IEditVerifyHandler can be attached to ITextSpinButton objects.

The Motif XmText widget does not support embedded NULLs or embedded nonprintable characters. Only the text preceding the first NULL is loaded into the control.

For a portable application, do not load text containing embedded NULLs or nonprintable characters into an IMultiLineEdit object. Additionally, remember that end-of-line characters are different on different systems.

Public Functions

Change Operations

Use these members to track changes for the MLE.

hasTextChanged

If any changes have been made to the MLE since the last time the changed flag was reset, true is returned. Otherwise, false is returned.

```
Boolean
  hasTextChanged() const;
```

Win	PM	Motif
Y	*Y*	*Y*

resetTextChangedFlag

Resets the changed flag so that from this point forward changes to the MLE can be detected. This is the same as IMultiLineEdit::setTextChangedFlag(false).

```
virtual IMultiLineEdit&
  resetTextChangedFlag();
```

Win	PM	Motif
Y	*Y*	*I*

© IBM Corp. 1992, 1996

setTextChangedFlag

Sets a flag indicating the changed status of the MLE. If changed=true, a flag is set to indicate the MLE contents have changed. if changed=false, a flag is set to indicate the MLE contents have not changed.

```
virtual IMultiLineEdit&
  setTextChangedFlag( Boolean changed = true );
```

Win	PM	Motif
Y	Y	I

Clipboard Operations

Use these members to transfer data between the clipboard and the multiline edit (MLE) control. Each of these operations deals with the selected text in the MLE control.

clear

Replaces the selected text in the MLE with blanks.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the MLE is selected before making the call to clear.

To remove the text contents of the MLE, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IMultiLineEdit&
  clear( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Exceptions	
IInvalidRequest	The MLE has no selected text.

copy

Copies the currently selected text from the MLE to the clipboard.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the MLE is selected before making the call to copy.

```
virtual IMultiLineEdit&
  copy( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Exceptions	
IInvalidRequest	The MLE has no selected text.
IAccessError	The operating system's request to copy from the MLE failed. See the exception text for further information about the failure.

cut Removes the currently selected text from the MLE control and places it in the clipboard.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the MLE is selected before making the call to cut.

To remove the text contents of the MLE, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IMultiLineEdit&
  cut( unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

PM The OS/2 release does not use the *timestamp* parameter.

Motif The optional parameter *timestamp* allows a time stamp value to be passed when this function is called from an event handler.

In AIX, get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Exceptions	
IInvalidRequest	The MLE has no selected text.
IAccessError	The operating system's request to cut from the MLE failed. See the exception text for further information about the failure.

discard Deletes all currently selected text.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the MLE is selected before making the call to discard.

© IBM Corp. 1992, 1996

To remove the text contents of the MLE, regardless of whether any text is selected, call setText, passing it a 0-length string, or removeAll.

```
virtual IMultiLineEdit&
  discard();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The MLE has no selected text.
IAccessError	The operating system's request to discard the selected text failed. See the exception text for further information about the failure.

paste

Inserts the contents of the clipboard into the MLE at the cursor position, replacing any selected text.

The user can select text or your code can call selectRange. The code can call hasSelectedText to determine if any text in the MLE is selected before making the call to paste.

```
virtual IMultiLineEdit&
  paste();
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

Exceptions	
IInvalidRequest	The clipboard does not contain any text.
IAccessError	The operating system's request to paste to the MLE failed. See the exception text for further information about the failure.

Colors

Use these members to query the current colors of the MLE.

backgroundColor

Returns the background color value of the MLE or the default if no color for the area has been set.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	Y	Y

Win Returns the default background color value of the MLE

foregroundColor

Returns the foreground color value of the MLE or the default if no color for the area has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	Y	Y

Win Returns the default foreground color value of the MLE

Compound Control

Use these members to access the top window system.

topHandle

Returns the top (that is, the oldest ancestor) window system control of the controls created by this class in its constructor or elsewhere. The User Interface Class Library uses the returned handle internally for, among other things, showing this object.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	N	Y

Constructors

You can construct and destruct objects of this class.

IMultiLineEdit

1

```
IMultiLineEdit( unsigned long id,
                IWindow* parent,
                IWindow* owner,
                const IRectangle& initial = IRectangle ( ),
                const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the parent window, owner window, optional size and location, and optional style parameters.

id An MLE control ID.

parent The parent window.

owner The owner window.

initial A rectangle for the control. It specifies the initial position and size of the object you are constructing. Optional.

style The initial style for the control. The default is classDefaultStyle (p. 665). Optional.

© IBM Corp. 1992, 1996

IMultiLineEdit

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2
```
IMultiLineEdit( unsigned long id,
                IWindow* parent );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the parent window.

id An MLE control ID.
parent The parent window.

3
```
IMultiLineEdit( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using the handle of an existing MLE.

handle The window handle of an existing MLE control.

~IMultiLineEdit

```
virtual
  ~IMultiLineEdit();
```

Win	PM	Motif
Y	Y	Y

Fonts

Use these members to modify the font in the MLE.

setFont Changes the font of the displayed text.

```
virtual IMultiLineEdit&
  setFont( const IFont& font );
```

Win	PM	Motif
N	Y	N

Exceptions	
IAccessError	The operating system's request to set the MLE font failed.

Import and Export Operations

Use these members to import and export text to a file.

exportSelectedTextToFile

Saves the currently marked text to the specified file and returns the number of bytes written to the file.

fileName The name of the file you want the text saved to.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual unsigned long
  exportSelectedTextToFile( const char* fileName,
                            EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX only supports the noTran format enumerator.

Exceptions	
IInvalidRequest	The MLE has no selected text.

exportToFile

Saves the contents of the MLE to the specified file and returns the number of bytes written to the file. If the file cannot be opened, zero is returned.

fileName The name of the file you want the text saved to.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual unsigned long
  exportToFile( const char* fileName,
                EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX only supports the noTran format enumerator.

importFromFile

Inserts the contents of the specified file into the MLE at the current cursor position and returns the number of bytes imported.

fileName The name of the file you want the text imported from.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual unsigned long
  importFromFile( const char* fileName,
                  EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

Exceptions	
IAccessError	An attempt to open or read from a file has failed.

Line Operations

These members use the terms *line* and *line number* as the arguments or the returns from these members. A *line* is a line on the display after the application of word-wrap. It does not mean a line as defined by the CR-LF line-break sequence.

addLine

Inserts the specified NULL-terminated text at the specified line number.

text The text you want to insert into the MLE.

lineNumber
The line number where you want to insert the text.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual IMultiLineEdit&
  addLine( const char* text,
           unsigned long lineNumber,
           EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

addLineAsLast

Inserts the specified NULL-terminated text as the last line in the MLE.

text The text you want to insert into the MLE.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual IMultiLineEdit&
  addLineAsLast( const char* text,
                 EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

cursorLinePosition

Returns the line number of the line that contains the cursor.

```
unsigned long
  cursorLinePosition() const;
```

Win	PM	Motif
Y	*Y*	*N*

cursorPosition

Returns the character position from the start of the MLE to the current cursor location.

```
unsigned long
  cursorPosition() const;
```

Win	PM	Motif
Y	*Y*	*N*

numberOfLines

Returns the number of lines in the MLE, including word breaks. The number of lines is based on the scrollable size of the MLE. You must set the size of the MLE before using this function. Otherwise, the returned value is not correct.

```
unsigned long
  numberOfLines() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

removeLine

Deletes the specified line of text.

```
virtual IMultiLineEdit&
  removeLine( unsigned long lineNumber );
```

Win	PM	Motif
Y	*Y*	*Y*

Motif If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

setCursorLinePosition

Moves the cursor position to the first position of the specified line.

```
virtual IMultiLineEdit&
  setCursorLinePosition( unsigned long lineNumber );
```

Win	PM	Motif
Y	*Y*	*N*

Motif If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

© IBM Corp. 1992, 1996

setCursorPosition

Moves the cursor to a specific position in the MLE. The count begins at the first position in the MLE, not at the cursor's current position.

```
virtual IMultiLineEdit&
  setCursorPosition( unsigned long cursorPosition );
```

Win	PM	Motif
Y	Y	N

setTop

Makes the specified line the topmost visible line on the screen.

```
virtual IMultiLineEdit&
  setTop( unsigned long lineNumber );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to set the top line of the MLE failed.

top

Returns the line number of the line currently visible at the top of the screen.

```
unsigned long
  top() const;
```

Win	PM	Motif
Y	Y	Y

visibleLines

Returns the number of lines that completely fits inside the edit region of the MLE. This value is based on the current font.

```
unsigned long
  visibleLines() const;
```

Win	PM	Motif
Y	Y	Y

Motif If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables or disables the MLE control to send notifications to any observer objects.

```
virtual IMultiLineEdit&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Read-Only Operations

Use these members to set and query the read-only style.

disableDataUpdate

Prevents inserting or changing characters in the MLE's text.

```
virtual IMultiLineEdit&
  disableDataUpdate();
```

Win	PM	Motif
Y	Y	Y

enableDataUpdate

Enables or disables the read-only mode of the MLE control.

```
virtual IMultiLineEdit&
  enableDataUpdate( Boolean update = true );
```

Win	PM	Motif
Y	Y	Y

isWriteable

If the contents of the MLE can be modified, true is returned. Otherwise, false is returned.

```
Boolean
  isWriteable() const;
```

Win	PM	Motif
Y	Y	Y

Refresh Operations

Use these members to enable and disable screen updates.

disableUpdate

Disables screen updates. Use this function when your application makes changes throughout an MLE. Doing so avoids unnecessary overhead caused by attempts to keep the screen display updated. While update is disabled, mouse and keyboard messages are processed by beeping and ignoring them. The mouse pointer changes to the standard system "wait" pointer.

```
virtual IMultiLineEdit&
  disableUpdate();
```

Win	PM	Motif
Y	Y	Y

Motif Enabling and disabling screen refreshes is not typically done in Motif because screen refresh management is handled automatically by X. However, to ensure the portability of logic that requires disabling of input temporarily whenever refresh is disabled, the User Interface Class Library disables input in Motif, also. Screen refresh is always enabled.

enableUpdate

Enables screen updates.

© IBM Corp. 1992, 1996

```
virtual IMultiLineEdit&
  enableUpdate( Boolean update = true );
```

Win	PM	Motif
Y	Y	Y

Motif Enabling and disabling screen refreshes is not typically done in Motif, because screen refresh management is handled automatically by X. However, to ensure the portability of logic that requires disabling of input temporarily whenever refresh is disabled, the User Interface Class Library disables input in Motif, also.

Selected Text

Use these members to access the currently selected text of the MLE.

hasSelectedText

If any of the MLE's text is selected, true is returned. Otherwise, false is returned.

```
virtual Boolean
  hasSelectedText() const;
```

Win	PM	Motif
Y	Y	Y

selectedRange

Returns the range of the selected text. If no text is selected, an exception is thrown. The selected range is the index of the first character selected and the index of the last character selected. The index is 0-based.

```
virtual IRange
  selectedRange() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The MLE has no selected text.

selectedText

Returns the selected text string. No exception is thrown if there is no text selected; a NULL IString is returned.

```
virtual IString
  selectedText() const;
```

Win	PM	Motif
Y	Y	Y

selectedTextLength

Returns the size of the selected area, in bytes. The length includes CR-LFs, but not NULL terminators. No exception is thrown if there is no text selected; 0 is returned.

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

Win	PM	Motif
Y	Y	Y

selectRange

Selects a range of text.

range The selected range is the index of the first character selected (or to be selected) and the index of the last character selected (or to be selected). The index is 0-based. If you do not specify *range*, the default selects all of the text. If you specify the *range* to be *IMultiLineEdit::deselect*, then the multi line edit will be deselected.

timestamp In AIX, a value within the time member of the XEvent structure.

```
virtual IMultiLineEdit&
  selectRange( const IRange& range = IRange ( 0 , end ),
               unsigned long timestamp = 0 );
```

Win	PM	Motif
Y	Y	Y

Size Operations

Use these members to set and query the edit region.

editRegionHeight

Returns the height of the edit region.

```
unsigned long
  editRegionHeight() const;
```

Win	PM	Motif
Y	Y	Y

editRegionWidth

Returns the width of the edit region.

```
unsigned long
  editRegionWidth() const;
```

Win	PM	Motif
Y	Y	Y

setEditRegion

Sets the size of the edit region so that it covers the entire MLE window. The edit region is the rectangular area in which the text is displayed.

1

```
virtual IMultiLineEdit&
  setEditRegion();
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, if the window is too small to show all of the text, it is not an error. In this situation, this function does not throw an exception.

Exceptions	
IAccessError	The operating system's request to set the region failed.

© IBM Corp. 1992, 1996

2 `virtual IMultiLineEdit&`
`  setEditRegion( const ISize& sizeEditRegion );`

Win	PM	Motif
Y	Y	Y

Motif In Motif, if the window is too small to show all of the text, it is not an error. In this situation, this function does not throw an exception.

Exceptions	
IInvalidParameter	The operating system's request to set the region width and height failed. The sizeEditRegion parameter is invalid.

setEditRegionHeight

Sets the height of the edit region.

```
virtual IMultiLineEdit&
  setEditRegionHeight( long height );
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, if the window is too small to show all of the text, it is not an error. In this situation, this function does not throw an exception.

Exceptions	
IInvalidParameter	The operating system's request to set the region height failed. The height parameter is invalid.

setEditRegionWidth

Sets the width of the edit region.

```
virtual IMultiLineEdit&
  setEditRegionWidth( long width );
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, if the window is too small to show all of the text, it is not an error. In this situation, this function does not throw an exception.

Exceptions	
IInvalidParameter	The operating system's request to set the region width failed. The width parameter is invalid.

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the MLE styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 665) unless you have changed the style using setDefaultStyle (p. 657).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent MLEs.

style Use the styles provided by IMultiLineEdit Styles (p. 664) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Tab Stops

Use these members to manipulate characteristics of the tab operation.

setTab

Sets the number of pixels between tab stops inside the MLE.

```
virtual IMultiLineEdit&
  setTab( unsigned long tabPixelInterval );
```

Win	PM	Motif
Y	Y	I

Exceptions	
IAccessError	The operating system's request to set the tab failed.

Text Operations

Use these members to access the text of the MLE.

© IBM Corp. 1992, 1996

IMultiLineEdit

add

Inserts the specified text into the MLE at the current cursor position and positions the cursor at the end of the inserted text.

Note: This function strips the last CR-LF characters. Therefore, if you need to explicitly use CR-LF, do one of the following:

- Use CR-LF at the beginning of an input string.
- Use IMultiLineEdit::addLine (p. 650).

text The text you want to insert into the MLE.

textSize If you want to add text containing more than one NULL character, specify *textSize*. If you use the default, this function adds the characters up to the first NULL character.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual IMultiLineEdit&
  add( const char* text,
       unsigned long textSize = 0,
       EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

addAsLast

Inserts the specified text into the MLE at the end of the current text but does not change the cursor position.

text The text you want to insert into the MLE.

textSize If you want to add text containing more than one NULL character, specify *textSize*. If you use the default, this function adds the characters up to the first NULL character.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual IMultiLineEdit&
  addAsLast( const char* text,
             unsigned long textSize = 0,
             EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

addAtOffset

Inserts the specified text into the MLE at the point specified. At the conclusion of this insert, this function positions the cursor at the same character as it was before the insert.

text The text you want to insert into the MLE.

charnumber
The number of the character after which you want to insert *text*. This is a 0-based index.

textSize If you want to add text containing more than one NULL character, specify *textSize*. If you use the default, this function adds the characters up to the first NULL character.

type Use the enumeration EOLFormat (p. 666) to specify the type of format. The default is cfText.

```
virtual IMultiLineEdit&
  addAtOffset( const char* text,
               unsigned long offset,
               unsigned long textSize = 0,
               EOLFormat type = cfText );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

limit

Returns the currently set number of bytes that the MLE can hold.

```
unsigned long
  limit() const;
```

Win	PM	Motif
Y	Y	Y

setLimit

Sets a limit on the number of bytes that can be input into the MLE.

```
virtual IMultiLineEdit&
  setLimit( unsigned long newLimit );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to set the text limit failed, or the MLE already contains more text than the proposed limit.

setText

Overwrites the current contents of the MLE with the specified text.

buffer The input buffer containing the new text. The input buffer can contain NULL characters.

© IBM Corp. 1992, 1996

1
```
virtual IMultiLineEdit&
  setText( const char* buffer,
           unsigned long bufferSize );
```
Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL.

2
```
virtual IMultiLineEdit&
  setText( const char* text );
```
Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL.

3
```
virtual IMultiLineEdit&
  setText( const IResourceId& text );
```
Win	PM	Motif
Y	Y	Y

Motif AIX does not support embedded NULLs. This function only loads the text up to the first NULL.

text If you do not specify *lineNumber*, this function returns the current contents of the MLE. If you do specify *lineNumber*, this function returns the current MLE text at the specified line.

1
```
virtual IString
  text( unsigned long lineNumber ) const;
```
Win	PM	Motif
Y	Y	Y

2
```
virtual IString
  text() const;
```
Win	PM	Motif
Y	Y	Y

textLength Returns the current length of the MLE text, in bytes. The length includes any CR-LFs present.

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

Text Removal

These members remove text from the MLE.

removeAll Deletes the entire contents of the MLE.

```
virtual IMultiLineEdit&
  removeAll();
```
Win	PM	Motif
Y	Y	Y

Undo Operations

These members provide undo capabilities.

isUndoable If any undoable actions have been performed on the contents of the MLE, true is returned. Otherwise, false is returned.

```
Boolean
  isUndoable() const;
```

Win	PM	Motif
Y	*Y*	*I*

Motif This function always returns false.

undo Restores the MLE's content to the state it was in before the last change. If the undo fails, the change flag is reset.

```
virtual IMultiLineEdit&
  undo();
```

Win	PM	Motif
Y	*Y*	*I*

Word-Wrap Operations

These members provide for word-wrap in the MLE.

disableWordWrap

Disables word-wrap mode in the MLE.

```
virtual IMultiLineEdit&
  disableWordWrap();
```

Win	PM	Motif
I	*Y*	*Y*

enableWordWrap

Enables word-wrap mode in the MLE. Word-wrap breaks the text at word boundaries to keep all of the text in a line visible, even if the text is wider than the MLE's window. The text is still one line of text. No carriage-control or line-feed characters are added.

```
virtual IMultiLineEdit&
  enableWordWrap( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*Y*

Exceptions	
IAccessError	The operating system's request to enable word-wrapping failed.

isWordWrap If the MLE is in word-wrap mode, true is returned. Otherwise, false is returned.

© IBM Corp. 1992, 1996

IMultiLineEdit

```
Boolean
  isWordWrap() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Drag and Drop Support

Use these members to customize the conditions for detecting a drag operation.

isDragStarting

Request the start of a drag operation. A drag operation will occur if the MLE contains selected text and the mouse pointer was over that selection area when the button was pressed.

```
virtual Boolean
  isDragStarting( IEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

passEventToOwner

Determines if the event is passed on to the owner.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	*N*	*Y*

registerCallbacks

Registers all possible callbacks to this object for events it might receive. IHandler-derived classes later determine which events they will process.

If classes you derive override registerCallbacks, the override must call Inherited::registerCallbacks to register the applicable callbacks.

This class does not add any X event handlers.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes any callbacks and X event handlers added by registerCallbacks.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

dataUpdateId Notification identifier provided to observers when the data update mode of the MLE changes. IMultiLineEdit provides a Boolean value in the eventData field of the INotificationEvent. This value is true if data update is now enabled, and false if data update is disabled.

```
static INotificationId const
  dataUpdateId;
```

Win	PM	Motif
Y	Y	Y

limitId Notification identifier provided to observers when the text limit of the MLE changes. IMultiLineEdit provides the new text limit value in the eventData field of the INotificationEvent.

```
static INotificationId const
  limitId;
```

Win	PM	Motif
Y	Y	Y

Selected Text

Use these members to access the currently selected text of the MLE.

deselect Denotes that the text should be deselected when calling IMultiLineEdit::selectRange.

```
static const long
  deselect;
```

Win	PM	Motif
Y	Y	Y

end Denotes the end of the text for selecting text.

```
static const long
  end;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the MLE styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

border Causes a thin border to be drawn around the MLE.

```
static const Style
  border;
```

Win	PM	Motif
Y	*Y*	*Y*

border3D Adds an etched 3D border to the control. This style is ignored if border is not specified.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	*I*	*I*

Win This style is ignored on Windows NT 3.51 with Program Manager.

W32s This style is ignored on Win32s.

classDefaultStyle

Provides the original default style for this class, which is the following: IMultiLineEdit::border | IMultiLineEdit::verticalScroll | IMultiLineEdit::wordWrap | IMultiLineEdit::border3D | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

horizontalScroll

Adds a horizontal scroll bar to the MLE.

```
static const Style
  horizontalScroll;
```

Win	PM	Motif
Y	*Y*	*Y*

ignoreTab Causes the MLE to ignore tab keystrokes. The function sends a keyboard event to the owner of the MLE.

```
static const Style
  ignoreTab;
```

Win	PM	Motif
Y	*Y*	*Y*

readOnly Restricts the user from entering any data.

```
static const Style
  readOnly;
```

Win	PM	Motif
Y	*Y*	*Y*

verticalScroll Adds a vertical scroll bar to the MLE.

```
static const Style
  verticalScroll;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

wordWrap Causes text word-wrap at the end of a line.

```
static const Style
  wordWrap;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IMultiLineEdit contains the following nested classes:

IMultiLineEdit::Style (see page 668)

```
EOLFormat {
    cfText,
    noTran,
    MLEFormat
    };
```

Use these enumerators to specify the format for the end-of-line characters when importing and exporting text:

cfText
Ends each line with a carriage-return or line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.

noTran
Uses LF for line delineation and guarantees any text imported into the MLE in this format can be recovered in exactly the same form when exported.

MLEFormat
On import, recognizes CR-LF as denoting hard line-breaks and ignores the sequence CR-CR-LF. On export, uses CR-LF to denote a hard line-break and CR-CR-LF to denote a soft line-break caused by word-wrapping.

Win The noTran enumerator is not supported in Windows. The MLEFormat enumerator is used instead.

Motif AIX only supports the noTran format enumerator. If you specify any other EOLFormat (p. 666), the member functions ignore it and use noTran instead.

© IBM Corp. 1992, 1996

IMultiLineEdit::Style

Derivation

IBase
IBitFlag
IMultiLineEdit::Style

Inherited By None.

Header File imle.hpp

The nested class IMultiLineEdit::Style provides a set of valid styles for the IMultiLineEdit (p. 641) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IMultiLineEditNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          IMultiLineEditNotifyHandler

Inherited By None.

Header File imlenhdr.hpp

Members

Member	Page
Constructor	669
dispatchHandlerEvent	670
~IMultiLineEditNotifyHandler	670

The IMultiLineEditNotifyHandler class processes events for all classes of multiline edit (MLE) controls.

This class is designed to handle events that require the IMultiLineEdit class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IMultiLineEditNotifyHandler

This is the default constructor and accepts no parameters.

```
IMultiLineEditNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

IMultiLineEditNotifyHandler

~IMultiLineEditNotifyHandler

```
virtual
  ~IMultiLineEditNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the MLE observers if any of the following events are received:

- dataUpdate event
- limit event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

ITextControlNotifyHandler		
dispatchHandlerEvent		

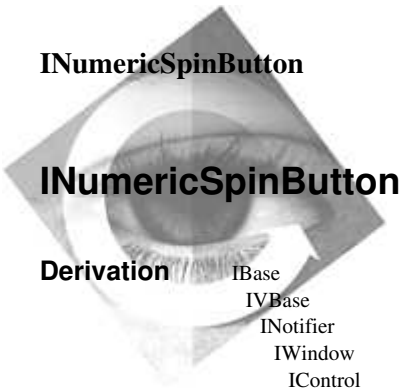
IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



INumericSpinButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
IBaseSpinButton
INumericSpinButton

Inherited By None.

Header File ispinnum.hpp

Members

Member	Page	Member	Page
Constructor	673	setLimit	674
classDefaultStyle	679	setRange	676
convertToGUIStyle	677	setValue	676
defaultStyle	677	spinDown	675
enableNotification	675	spinTo	676
isSpinFieldValid	675	spinUp	675
padWithZeros	680	unregisterCallbacks	679
range	675	value	676
registerCallbacks	678	~INumericSpinButton	674
setDefaultStyle	677		

The INumericSpinButton class creates and manages numeric spin button control windows.

Handlers derived from the following classes can handle events for INumericSpinButton objects:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISpinHandler (p. 896)

Motif The INumericSpinButton constructor creates objects of this class using the following Motif widgets:

- An XmForm widget is created with a single line XmText child.

© Copyright IBM Corp. 1992, 1996

- If the spin button has the style IBaseSpinButton::master, an XmForm widget containing two XmArrowButton widgets is also created as a child of the top XmForm. IWindow::handle returns the handle of the XmText widget.

The User Interface Class Library provides the behavior of an INumericSpinButton object via private callbacks and a default handler. The INumericSpinButton class uses a default handler derived from the class IKeyboardHandler. Therefore, attach user-defined handlers derived from IKeyboardHandler to the INumericSpinButton object rather than to its owner window. Doing so enables events to be dispatched to user-defined handlers before the default handler.

In addition, handlers derived from IEditVerifyHandler can be attached to ITextSpinButton objects.

The AIX environment supports only the first constructor, which creates an object of this class from a control ID, parent and owner windows, a rectangle, and a style.

Public Functions

Constructors

You can construct and destruct objects of this class.

INumericSpinButton

1
```
INumericSpinButton(
   unsigned long id,
   IWindow* parent,
   IWindow* owner,
   const IRectangle& initial = IRectangle ( ),
   const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by creating the specified numeric spin button control and an object for it. The numeric spin button has an initial value of 0 and an initial range of (0,0).

windowId A unique ID for the numeric spin button control.

parent The parent window.

owner The owner window.

initial A rectangle defining the size and placement of the numeric spin button window. Optional.

style The style of the control. Optional.

© IBM Corp. 1992, 1996

INumericSpinButton

2 `INumericSpinButton( unsigned long id, IWindow* parent );`

Win	PM	Motif
Y	Y	N

You can construct objects of this class by creating an object for the specified numeric spin button control.

id A unique ID for the numeric spin button control.
parent The parent dialog window.

Win This constructor can only be used to create an object with the pmCompatible (p. 118) style.

Exceptions	
IInvalidRequest	The ID must be for a numeric-only spin button control.

3 `INumericSpinButton( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	N

You can construct objects of this class by creating an object for the specified numeric spin button control.

handle The window handle of an existing numeric spin button control.

Win This constructor can only be used to create an object with the pmCompatible (p. 118) style.

Exceptions	
IInvalidRequest	The handle must be for a numeric-only spin button control.

~INumericSpinButton

```
virtual
  ~INumericSpinButton();
```

Win	PM	Motif
Y	Y	Y

Limit and Spin

Use these members to manage the spin field for objects of this class.

setLimit Sets the number of characters permitted in the spin field. The User Interface Class Library defines this limit as 255 at the time of construction.

Note: If you call this function with a limit that does not display all the numbers in the spin button range, erratic results can occur when the button is spun. For example, if the range is 1 to 100 and the limit is set to 2, the button spins up

to 99 and then spins to 10 instead of 100. Spinning down, the button wraps from 1 to 10.

```
virtual INumericSpinButton&
  setLimit( unsigned long aNumber );
```

Win	PM	Motif
N	N	Y

spinDown Spins the button down the specified number of times.

```
virtual INumericSpinButton&
  spinDown( unsigned long spinBy = 1 );
```

Win	PM	Motif
N	N	Y

spinUp Spins the button up the specified number of times.

```
virtual INumericSpinButton&
  spinUp( unsigned long spinBy = 1 );
```

Win	PM	Motif
N	N	Y

Notification Members

Use these members to enable notifications sent to observer objects.

enableNotification

Causes the numeric spin button control to send notifications to any observer objects added.

```
virtual INumericSpinButton&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Number Range Operations

Use these members to query, modify, and validate the spin field contents and range.

isSpinFieldValid

If the contents of the spin field are within the number range, true is returned. This function ignores the *caseSensitive* flag for numeric spin buttons.

```
virtual Boolean
  isSpinFieldValid( Boolean caseSensitive = false ) const;
```

Win	PM	Motif
Y	Y	Y

range Returns the number range of the spin button.

```
virtual IRange
  range() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The operating system's request to query the spin button's limits has failed.

setRange

Sets the lower and upper limit of the spin button range.

aNewRange
The new lower and upper limit.

override If you set *override* to true and the current value in the spin field is outside the new range, the current value in the spin field does not change. If *override* is false and the current value in the spin field is outside the new range, the current spin value in the spin field is set to the limit closest to the value. The default is false.

```
virtual INumericSpinButton&
  setRange( const IRange& aNewRange,
            Boolean override = false );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The new range is invalid.

setValue

Sets the displayed value of the spin field, regardless of the validity of the number.

```
virtual INumericSpinButton&
  setValue( long aValue );
```

Win	PM	Motif
Y	Y	Y

spinTo

Spins the spin button until the specified value displays. If the specified value is out of range, one of the following happens:

- If *spinToClosest* is true, the button spins to the closest limit.
- If *spinToClosest* is false, an IInvalidParameter exception is thrown.

```
virtual INumericSpinButton&
  spinTo( long aValue,
          Boolean spinToClosest = false );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The spin value is not valid.

value

Returns the current value displayed in the spin field.

```
virtual long
  value() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the numeric spin button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IBaseSpinButton Styles (p. 116)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 679) unless you have changed the style using setDefaultStyle (p. 677).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent numeric spin buttons.

style Use the styles provided by INumericSpinButton Styles (p. 679) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IBaseSpinButton		
addBorder	hasBorder	resetBackgroundColor
alignment	isFastSpinEnabled	resetForegroundColor
backgroundColor	isMaster	setAlignment

© IBM Corp. 1992, 1996

INumericSpinButton

IBaseSpinButton		
convertToGUIStyle	isPMCompatible	setBackgroundColor
disableDataUpdate	isServant	setForegroundColor
disableFastSpin	isSpinFieldValid	setLimit
enable	isWriteable	setMaster
enableDataUpdate	limit	spinDown
enableFastSpin	moveSizeTo	spinUp
foregroundColor	removeBorder	topHandle

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Registers the Motif callbacks for the underlying widgets. It is called during construction of the INumericSpinButton window.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

unregisterCallbacks

Unregisters the Motif callbacks for the underlying widgets. It is called during destruction of the INumericSpinButton window.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Inherited Protected Functions

IBaseSpinButton		
calcMinimumSize	initialize	registerCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the numeric spin button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IBaseSpinButton Styles (p. 116)

classDefaultStyle

Provides the original default style for this class, which is the following: IBaseSpinButton::master | IBaseSpinButton::leftAlign | IBaseSpinButton::border3D | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

INumericSpinButton

padWithZeros

Pads the displayed number with zeros at the front.

```
static const Style
  padWithZeros;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

IBaseSpinButton		
border3D	master	rightAlign
centerAlign	noBorder	servant
fastSpin	pmCompatible	textId
leftAlign	readOnly	valueId

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

INumericSpinButton contains the following nested classes:

INumericSpinButton::Style (see page 682)

© IBM Corp. 1992, 1996

INumericSpinButton::Style

Derivation

IBase
IBitFlag
INumericSpinButton::Style

Inherited By None.

Header File ispinnum.hpp

The nested class INumericSpinButton::Style provides a set of valid styles for the INumericSpinButton (p. 672) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

INumericSpinButtonNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        INumericSpinButtonNotifyHandler

Inherited By None.

Header File ispbtnnh.hpp

Members

Member	Page
Constructor	683
dispatchHandlerEvent	684
~INumericSpinButtonNotifyHandler	683

The INumericSpinButtonNotifyHandler class processes events for all classes of numeric spin buttons.

This class is designed to handle events that require the numeric spin button class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

INumericSpinButtonNotifyHandler

This is the default constructor and accepts no parameters.

```
INumericSpinButtonNotifyHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

~INumericSpinButtonNotifyHandler

© Copyright IBM Corp. 1992, 1996

INumericSpinButtonNotifyHandler

```
virtual
  ~INumericSpinButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the numeric spin button observers if the following event is received:

- spin button change event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

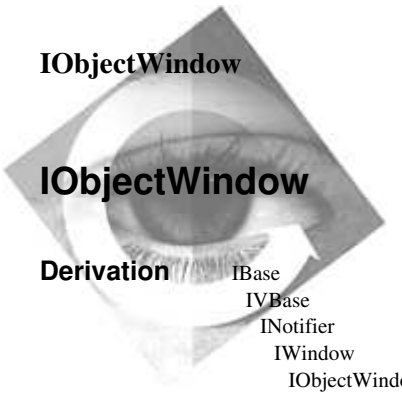
IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IObjectWindow

Derivation

IBase
IVBase
INotifier
IWindow
IObjectWindow

Inherited By None.

Header File iobjwin.hpp

Members

Member	Page
Constructor	687
close	687
~IObjectWindow	687

The IObjectWindow class represents object windows. *Object windows* are invisible windows that process window events. Essentially, these windows receive only messages that are explicitly sent to them.

One possible use of this class is to process a set of user-defined messages.

PM These windows have minimal system overhead because they are created with a parent of HWND_OBJECT.

Win IObjectWindow creates an invisible window. The parent of the window is the window returned by the IWindow::objectWindow (p. 1051) function.

Motif IObjectWindow uses the WMShell widget with the XmNoverrideRedirect resource set to true. This resource enables the window manager to ignore the shell widget but the widget is still able to process events.

Public Functions

Closing

After you construct an object window, it exists until you close it or you end the thread with its message queue.

© Copyright IBM Corp. 1992, 1996

close Closes the object window.

The User Interface Class Library considers an IObjectWindow object to represent a primary window, unless you call IWindow::setParent (p. 1070). The User Interface Class Library automatically ends the message processing for a thread when all of its primary windows are closed.

```
IObjectWindow&
  close();
```

Win	PM	Motif
Y	Y	N

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IObjectWindow objects because both the copy constructor and assignment operator are private functions.

IObjectWindow

```
IObjectWindow();
```

Win	PM	Motif
Y	Y	Y

Default constructor.

Exceptions	
IAccessError	The system could not register the object window class.

~IObjectWindow

```
virtual
  ~IObjectWindow();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IObjectWindow

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IOutlineBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IOutlineBox

Inherited By None.

Header File ioutlbox.hpp

Members

Member	Page	Member	Page
Constructor	690	moveSizeTo	693
backgroundFrame	694	outlineType	691
border3D	694	position	693
classDefaultStyle	694	setDefaultStyle	692
convertToGUIStyle	692	setForegroundColor	690
defaultStyle	692	setOutlineType	692
foregroundColor	689	size	693
foregroundFrame	695	visibleRectangle	691
halftoneFrame	695	~IOutlineBox	691

The IOutlineBox class creates and manages an unfilled rectangle control.

Public Functions

Colors

Use these members to query and set the color of the outline box.

foregroundColor

Returns the foreground color value of the outline box or the default if no color for the outline box has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Win Returns the default foreground color value of the outline box.

© Copyright IBM Corp. 1992, 1996

setForegroundColor

Sets the foreground color of the outline box.

```
virtual IOutlineBox&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
N	N	Y

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IOutlineBox objects because both the copy constructor and assignment operator are private functions.

IOutlineBox

1

```
IOutlineBox( unsigned long id,
             IWindow* parent,
             IWindow* owner,
             const IRectangle& initial = IRectangle ( ),
             const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Use this constructor to create an outline box control and object from a control ID, parent and owner windows, a rectangle, and a style.

id An outline box control ID.

parent The parent window.

owner The owner window.

initial A rectangle for the outline box control. It specifies the initial position and size of the outline box you are constructing. Optional.

style The initial style for the outline box control. The default is classDefaultStyle (p. 694). Optional.

Exceptions	
IInvalidParameter	A specified parameter was invalid. There are two possible causes. The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent. Or, a valid outline box style was not specified. You must include one of the following in the outline box style: backgroundFrame, foregroundFrame, or halftoneFrame.

2

```
IOutlineBox( unsigned long id,
             IWindow* parent );
```

Win	PM	Motif
Y	Y	N

Use this constructor to create an IOutlineBox object for an outline box control on a dialog window.

id An outline box control ID.
parent The parent dialog window.

3 `IOutlineBox( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	N

Use this constructor to create an IOutlineBox object from an existing outline box control.

handle An existing outline box window handle.

~IOutlineBox

```
virtual
  ~IOutlineBox();
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

visibleRectangle

Returns a rectangle that represents the area of the window that is actually painted by the control.

```
virtual IRectangle
  visibleRectangle() const;
```

Win	PM	Motif
Y	Y	N

Outline Types

Use these members to query and set the outline type. You can specify one of three types of outline box. You specify that the outline box be in the currently set foreground color, the system background color, or halftone shading using the foreground color.

outlineType

Returns the current type of outline for this outline box object. The returned value is one of the enumerators provided by OutlineType (p. 696).

```
OutlineType
  outlineType() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

setOutlineType

Sets the type of outline for this outline box object.

```
IOutlineBox&
  setOutlineType( const OutlineType& type );
```

Win	PM	Motif
Y	Y	Y

type Enumerated type of outline. Use the enumerators provided by OutlineType (p. 696) to specify the type of outline to set.

Exceptions	
IInvalidParameter	An invalid OutlineType was specified. You must specify one of the valid OutlineType values. Current valid values are foreground, background, and halftone.

Styles

These style members provide a set of valid outline box styles for this class. Use these members to query and set the outline box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, will be returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 694) unless you have changed the style using setDefaultStyle (p. 692).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent outline boxes.

style Use the styles provided by IOutlineBox Styles (p. 694) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

style Use the styles provided by IOutlineBox::Styles (p. 694) to specify the default style.

Window Positioning

Use these members to set and query the size and position of the outline box.

moveSizeTo Changes the position and size of the outline box.

```
virtual IOutlineBox&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	N	Y

position Returns the position of the outline box.

```
virtual IPoint
  position() const;
```

Win	PM	Motif
N	N	Y

size Returns the size of the outline box.

```
virtual ISize
  size() const;
```

Win	PM	Motif
N	N	Y

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**

© IBM Corp. 1992, 1996

IBase		
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid outline box styles for this class. Use these members to query and set the outline box styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

backgroundFrame

Causes the outline box to have a frame equal to the background color.

```
static const Style
  backgroundFrame;
```

Win	PM	Motif
Y	*Y*	*Y*

border3D

Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	*I*	*I*

Win This style is ignored on Windows NT 3.51 with Program Manager.

W32s This style is ignored on Win32s.

classDefaultStyle

Provides the original default style for this class, which is the following:
IOutlineBox::foregroundFrame | IWindow::visible | IOutlineBox::border3D.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

foregroundFrame

Causes the outline box to have a frame equal to the foreground color.

```
static const Style
  foregroundFrame;
```

Win	PM	Motif
Y	*Y*	*Y*

halftoneFrame

Causes the outline box to have the frame shaded in a halftone of the foreground color.

```
static const Style
  halftoneFrame;
```

Win	PM	Motif
Y	*Y*	*I*

Motif The AIX release ignores the halftoneFrame.

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

Nested Classes

IOutlineBox contains the following nested classes:

IOutlineBox::Style (see page 697)

OutlineType

```
OutlineType {
    foreground,
    background,
    halftone
    };
```

Use these enumerators to specify outline types:

foreground
Specifies the currently set foreground color.

background
Specifies the system's background color.

halftone
Specifies halftone shading in the foreground color.

IOutlineBox::Style

Derivation

IBase
  IBitFlag
    IOutlineBox::Style

Inherited By None.

Header File ioutlbox.hpp

The nested class IOutlineBox::Style provides a set of valid styles for the IOutlineBox (p. 689) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IPaintConnectionTo

Derivation

IBase
IVBase
IHandler
IPaintHandler
IPaintConnectionTo

Inherited By None.

Header File ipainhdr.hpp

Members

Member	Page
Constructor	699
paintWindow	700
~IPaintConnectionTo	699

The IPaintConnectionTo class is a template class, derived from IPaintHandler (p. 706), that processes paint events. This class allows you to process paint events in objects that do not derive from IPaintHandler without having to manually derive from IPaintHandler, override the paintWindow (p. 708) function, and pass the event to the object.

To use the IPaintConnectionTo class to process paint events, follow these steps:

1. instantiate the IPaintConnectionTo template with a class containing a member function whose signature and behavior are the same as IPaintHandler::paintWindow (p. 708).
2. Construct an object of the new template class by passing the object and the address of the function handling the paint events.
3. Attach the template paint handler by using IHandler::handleEventsFor (p. 413) to pass the appropriate window to the paint handler.

You use an IPaintConnectionTo object anywhere you use an IPaintHandler object. See the class description of IPaintHandler (p. 706) for a description of the uses and limitations of these classes.

Public Functions

© Copyright IBM Corp. 1992, 1996

Constructors

You can construct and destruct objects of this class. You cannot copy or assign objects of this class.

IPaintConnectionTo

Constructs the IMouseConnectionTo object.

```
IPaintConnectionTo( ATarget& target,
                    MemberFunction memberFunction );
```

Win	PM	Motif
Y	N	N

The parameters are the following:

target The object to receive paint events.

memberFunction
The address of the member function to receive paint events.

~IPaintConnectionTo

```
virtual
  ~IPaintConnectionTo();
```

Win	PM	Motif
Y	N	N

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

© IBM Corp. 1992, 1996

Event Processing

These members are overridden to route paint events to the target object that you provided on construction.

paintWindow Overridden to route paint events to the member function of the Target object provided on construction.

```
virtual Boolean
  paintWindow( IPaintEvent& event );
```

Win	PM	Motif
Y	*N*	*N*

Inherited Protected Functions

IPaintHandler		
dispatchHandlerEvent	paintWindow	

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Type Definitions

MemberFunction

```
typedef Boolean (ATarget::*MemberFunction)(IPaintEvent&);
```

A pointer to a member function of Class ATarget that takes a reference to an IPaintEvent and returns a Boolean.

IPaintEvent

Derivation

IBase
IVBase
IEvent
IPaintEvent

Inherited By None.

Header File ipainevt.hpp

Members

Member	Page	Member	Page
Constructor	702	rect	704
clearBackground	702	setGraphicContext	704
drawText	703	~IPaintEvent	702
presSpaceHandle	704		

The IPaintEvent class represents an OS/2 Presentation Manager paint event, which requires a window or control to update its appearance on the screen. An object of IPaintHandler (p. 706) creates and passes this event to its virtual function IPaintHandler::paintWindow (p. 708) for processing. You must make all window updates using the presentation space handle supplied by the IPaintEvent object.

Motif Objects of the IPaintEvent class represent a Motif expose event.

As for all coordinates and rectangles in the User Interface Class Library, members of this class convert the rectangle that is part of this event from Presentation Manager coordinates (that is, lower-left origin-based coordinates) to Motif system coordinates (that is, upper-left origin-based coordinates).

When drawing, this class uses the presentation space (that is, graphic context) of the IWindow object for which this paint event was created. If you do not make all window updates using the presentation space handle supplied by the IPaintEvent object, you will not get the same colors and fonts as other sections of code painting in this window.

Public Functions

Constructors

You can construct and destruct objects of this class.

© Copyright IBM Corp. 1992, 1996

IPaintEvent

IPaintEvent

Constructs an IPaintEvent object from the specified event. IPaintHandler::dispatchHandlerEvent (p. 708) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function IPaintHandler::paintWindow (p. 708).

```
IPaintEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IPaintEvent

```
virtual
  ~IPaintEvent();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The clipping region could not be returned to its original state.
IAccessError	The clipping region could not be destroyed properly.

Painting

Use painting members to update a window's appearance on the screen.

clearBackground

Clears the specified portion of the window, filling it with the specified color.

1

```
IPaintEvent&
  clearBackground(
   const IColor& background =
     IGUIColor ( IGUIColor::desktopBgnd) );
```

Win	PM	Motif
Y	Y	Y

background

The background color to paint the window.

Use this function to clear the entire window.

Motif This function uses the screen rectangle for the exposed part of the window.

In X-Motif, the library uses the graphic context (GC) obtained via IWindow::presSpace (p. 1076). If you override this function, you can get the graphic context using the same function. You can then do the following:

- Change other attributes of the graphic context before calling the inherited function
- Paint the screen without changing the graphic context

If you use IWindow::presSpace, call IWindow::releasePresSpace (p. 1077) when you are done.

As for all coordinates and rectangles in the User Interface Class Library, this function converts the rectangle that is part of this event from Presentation Manager coordinates (that is, lower-left origin-based coordinates) to Motif system coordinates (that is, upper-left origin-based coordinates). This member function uses X Library drawing functions, such as XSetForeground, XClearArea, and XFillRectangle.

2

```
IPaintEvent&
  clearBackground(
   const IRectangle& fillRectangle,
   const IColor& background =
     IGUIColor ( IGUIColor::desktopBgnd ) );
```

Win	PM	Motif
Y	Y	Y

fillRectangle
The screen rectangle of the portion of the window to clear.

background
The background color to paint the *fillRectangle* portion of the window.

Use this function to clear only a portion of a window.

Motif In X-Motif, the library uses the graphic context (GC) obtained via IWindow::presSpace (p. 1076). If you override this function, you can get the graphic context using the same function. You can then do the following:

- Change other attributes of the graphic context before calling the inherited function
- Paint the screen without changing the graphic context

If you use presSpace, call IWindow::releasePresSpace (p. 1077) when you are done.

As for all coordinates and rectangles in the User Interface Class Library, this function converts the rectangle that is part of this event from Presentation Manager coordinates (that is, lower-left origin-based coordinates) to Motif system coordinates (that is, upper-left origin-based coordinates). This member function uses X Library drawing functions, such as XSetForeground, XClearArea, and XFillRectangle.

drawText Draws the specified text, beginning at the specified point, using the specified color for the text.

© IBM Corp. 1992, 1996

IPaintEvent

```
IPaintEvent&
  drawText(
   const char* text,
   const IPoint& atPoint,
   const IColor& textColor =
     IGUIColor ( IGUIColor::windowStaticText ) );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the library uses the graphic context (GC) obtained via IWindow::presSpace (p. 1076). If you override this function, you can get the graphic context using the same function. You can then do the following:

- Change other attributes of the graphic context before calling the inherited function
- Paint the screen without changing the graphic context

If you use IWindow::presSpace, call IWindow::releasePresSpace (p. 1077) when you are done.

This member function uses X Library drawing functions, such as XSetForeground, and XDrawString.

presSpaceHandle

Returns the handle of the presentation space to use for any drawing. This handle is obtained by calling IWindow::presSpace (p. 1076).

```
virtual IPresSpaceHandle
  presSpaceHandle();
```

Win	PM	Motif
Y	Y	Y

rect

Returns the screen rectangle for the invalidated part of the window.

```
IRectangle
  rect() const;
```

Win	PM	Motif
Y	Y	Y

setGraphicContext

Sets the graphic context used for drawing.

1
```
IPaintEvent&
  setGraphicContext( const IPresSpaceHandle& handle,
                     Boolean setClipRegion = true );
```

Win	PM	Motif
Y	Y	I

2
```
IPaintEvent&
  setGraphicContext( const IGraphicContext& context,
                     Boolean setClipRegion = true );
```

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

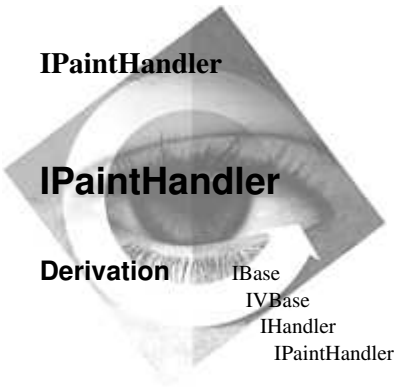
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IPaintHandler

Derivation

IBase
IVBase
IHandler
IPaintHandler

Inherited By IPaintConnectionTo

Header File ipainhdr.hpp

Members

Member	Page	Member	Page
Constructor	707	paintWindow	708
dispatchHandlerEvent	708	~IPaintHandler	707

The IPaintHandler class processes paint events for any window or control that requires the window or control to update its appearance on the screen.

Create a handler derived from IPaintHandler and attach it to a window or control. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the window or control to the paint handler.

When the paint handler receives a paint event, it creates an IPaintEvent (p. 701) object and routes that object to the IPaintHandler::paintWindow (p. 708) virtual function. Override this virtual function to supply your own specialized processing of a paint event.

The return value from the virtual function specifies whether the paint event is passed on for additional processing, as follows:

true The paint event requires no additional processing. Do not pass it to another handler.

false Pass the paint event to the next handler for additional processing, as follows:

- If there is another handler for the control or window, pass the paint event to the next handler.
- If this is the last handler for the control or window, call IWindow::defaultProcedure (p. 1082) to process the paint event.

Motif Objects of the IPaintEvent class represent a Motif expose event.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

Only derived classes can construct objects of this class.

~IPaintHandler

```
virtual
  ~IPaintHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

Only derived classes can construct objects of this class.

IPaintHandler

Derived classes call this default constructor to create objects of this class.

```
IPaintHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

IPaintHandler

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IPaintHandler object processes only window paint events.

dispatchHandlerEvent

If a paint event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

A resize handler contains event-processing members that you use to process a window paint event. Override at least one of these virtual functions in a derived class.

paintWindow

Implemented by derived classes to handle a paint event. A derived class must supply this function.

```
virtual Boolean
  paintWindow( IPaintEvent& event ) = 0;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IPointerHandle

Derivation

IBase
  IHandle
    IPointerHandle

Inherited By

ISystemPointerHandle

Header File

ihandle.hpp

Members

Member	Page	Member	Page
Constructor	709	operator Value	710
operator =	710	~IPointerHandle	710

The IPointerHandle class accesses and manages pointer resources for a pointing device, such as a mouse. IPointerHandle objects manage pointer resources through reference-counting. Reference-counting allows the system to use one bitmap (a pointer is a type of bitmap) in multiple places, and the library maintains the lifetime of this bitmap until all users are finished with it.

PM IPointerHandle is an alias for the OS/2 Programmer's Toolkit type HPOINTER.

Motif IPointerHandle is an alias for the XLib Pixmap type.

Public Functions

Constructors

You can construct, destruct, copy, and assign objects of this class. The copy constructor, assignment operator, and destructor track references to the associated pointer resource.

IPointerHandle

1 `IPointerHandle( Value hpointer = 0 );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class from a pointer handle (a value of type IHandle::Value), which defaults to 0.

© Copyright IBM Corp. 1992, 1996

IPointerHandle

2 `IPointerHandle( const IPointerHandle& aHandle );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class from an existing object using this copy constructor.

operator = Assigns the value of one pointer handle to another.

```
IPointerHandle&
  operator =( const IPointerHandle& aHandle );
```

Win	PM	Motif
Y	Y	Y

~IPointerHandle

```
~IPointerHandle();
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
N	N	Y

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

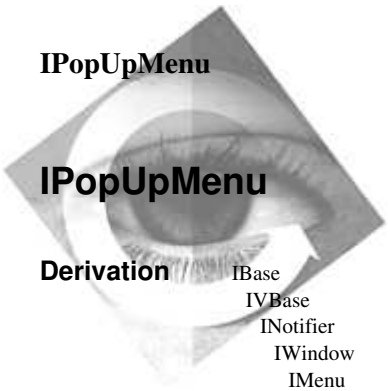
IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IPopUpMenu

Derivation

IBase
IVBase
INotifier
IWindow
IMenu
IPopUpMenu

Inherited By None.

Header File ipopmenu.hpp

Members

Member	Page	Member	Page
Constructor	713	topHandle	713
registerCallbacks	715	unregisterCallbacks	715
show	714	~IPopUpMenu	714

The IPopUpMenu class creates a pop-up menu. The User Interface Class Library supports the creation of pop-up menus on demand using IMenuHandler::makePopUpMenu (p. 578) and ICnrMenuHandler::makePopUpMenu (Vol. III).

ICommandEvents do not specify whether they are for a menu bar or pop-up menu. If you need to tell the difference, you must use different identifiers for the menu items to distinguish whether they are on the menu bar or the pop-up menu. For example, you might need to apply different actions to a menu item object when the user selects it from the menu bar as opposed to the pop-up menu.

Win To support menus loaded from a resource file, IPopUpMenu classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

Motif The User Interface Class Library adds a Motif callback routine to all pop-up menus to enable processing of the following menu events:

- Showing a menu on the display
- Selecting a menu item in a menu
- Removing a menu from the display

© Copyright IBM Corp. 1992, 1996

Public Functions

Compound Controls

Compound controls are GUI windows that are treated like primitive windows but, in fact, consist of two or more windows. User Interface Class Library uses compound control members to manipulate such windows.

topHandle Returns the menu shell (parent widget) of the pop-up menu. The IWindowHandle returned is used for positioning and managing the pop-up menu.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	*N*	*Y*

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IPopUpMenu objects because the copy constructor and assignment operator are private functions.

IPopUpMenu When you create a pop-up menu for an IContainerControl (Vol. III) or an IContainerObject (Vol. III) object, specify the container control as the owner of the pop-up menu. If you need to use an IMenuHandler (p. 576) or an ICnrMenuHandler (Vol. III), attach it to the container window, also.

1

```
IPopUpMenu( IWindow* owner,
            unsigned long menuWindowId,
            const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	*Y*	*Y*

Creates an empty pop-up menu using a window ID, owner window, and styles.

Exceptions	
IInvalidParameter	*owner* is 0. You must provide a valid owner window to create an IPopUpMenu object.

2

```
IPopUpMenu( const IResourceId& menuResId,
            IWindow* owner );
```

Win	PM	Motif
Y	*Y*	*Y*

Creates a pop-up menu object from a menu resource ID and window owner.

Win To support menus loaded from a resource file, IPopUpMenu classes have the following requirements:

Menu resources must be defined using the MENUEX keyword.

Unique IDs must be provided for all pop-up menu items, indicated by the POPUP keyword.

© IBM Corp. 1992, 1996

IPopUpMenu

~IPopUpMenu

Destroys the pop-up menu object.

```
virtual
  ~IPopUpMenu();
```

Win	PM	Motif
Y	Y	Y

Visibility

You cause the pop-up menu to become visible by specifying the location on the screen where it is to be placed.

show Displays the pop-up menu at the specified location. Specify *atLocation* using the window coordinates where you want the pop-up menu to appear. If you are showing the pop-up from within a menu handler, obtain the coordinates using IMenuEvent::mousePosition (p. 572). If you call IWindow::setAutoDeleteObject(true) (p. 1068), the IPopUpMenu object is deleted when the pop-up menu ends.

```
virtual IPopUpMenu&
  show( const IPoint& atLocation );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The pop-up menu could not be shown due to an operating system error.

Inherited Public Functions

IMenu		
add	enableNotification	resetDisabledBackgroundColor
addAsNext	foregroundColor	resetDisabledForegroundColor
addBitmap	hiliteBackgroundColor	resetForegroundColor
addItem	hiliteForegroundColor	resetHiliteBackgroundColor
addSeparator	id	resetHiliteForegroundColor
addSubmenu	isFrameWindow	selectItem
addText	isItemChecked	setBackgroundColor
backgroundColor	isItemEnabled	setBitmap
checkItem	isValid	setConditionalCascade
convertToGUIStyle	itemHelpId	**setDefaultStyle**
cursor	itemRect	setDisabledBackgroundColor
defaultStyle	menuHandle	setDisabledForegroundColor
deleteAt	menuItem	setForegroundColor

IMenu		
deleteItem	numberOfItems	setHiliteBackgroundColor
disabledBackgroundColor	owner	setHiliteForegroundColor
disabledForegroundColor	removeConditionalCascade	setItem
disableItem	removeSubmenu	setItemHelpId
elementAt	removeSubmenuAt	setSubmenu
enableItem	resetBackgroundColor	setText

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Adds X-Motif callbacks for the XmPopUpMenu row column widget.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes X-Motif callbacks from the XmPopUpMenu row column widget.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

© IBM Corp. 1992, 1996

IPopUpMenu

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Inherited Public Data

IMenu		
classDefaultStyle	**noStyle**	

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IPresSpaceHandle

Derivation

IBase
  IHandle
    IPresSpaceHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	717
operator _XGC *	718

The IPresSpaceHandle class accesses a presentation space.

PM IPresSpaceHandle is an alias for the OS/2 Programmer's Toolkit type HPS.

Motif IPresSpaceHandle is an alias for the GC (graphics context) type.

Public Functions

Constructors

You can construct objects of this class.

IPresSpaceHandle

1 `IPresSpaceHandle( Value hps = 0 );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class from a presentation space handle (a value of type IHandle::Value), which defaults to 0.

2 `IPresSpaceHandle( _XGC* hps );`

Win	PM	Motif
N	N	Y

You can construct objects of this class from a presentation space handle, a value of type X graphics context.

© Copyright IBM Corp. 1992, 1996

IPresSpaceHandle

3 `IPresSpaceHandle( int hps );`

Win	PM	Motif
N	*N*	*Y*

You can construct objects of this class from a presentation space handle, a value of type integer.

Operators

These members are the operators for this class.

operator _XGC *

Returns the handle value as an X graphics context.

```
operator _XGC *() 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**	

IPrivateResource

Derivation

IBase
  IVBase
    IResource
      IPrivateResource

Inherited By None.

Header File ireslock.hpp

Members

Member	Page
Constructor	720
handle	721
~IPrivateResource	720

The IPrivateResource class defines a resource that is used within a single process.

IResource (p. 805) and its inherited classes represent resources in the user's problem domain. Objects of these classes name a particular resource. For example, a user wants to serialize access to the data in a window list collection. The user creates an IResource inherited class to represent the window list and then uses IResourceLock (p. 820) (preferred) or a function of IResource itself to lock and unlock the resource to serialize the access. You can use this class either alone (lock and unlock) or in combination with an IResourceLock, whose constructor locks the resource and whose destructor unlocks the resource. If you need the lock only for a block of code, use IResourceLock. Use static objects because you only need a single object of a particular IResource.

The static object pointer discussions exist because there is no language-defined way to ensure that static objects are constructed when needed. Also, they are always constructed even if they are never needed. Combining a static pointer to an object with a static function to reference the object and then creating the object, if necessary, defers creating the object until it is needed.

Use this class for resources that are limited to the same process (as opposed to a public resource that is used by more than one process via its name).

You can use this class as a static key to serialize access to a private resource. You can also use this class as a mechanism to ensure that the static resource is constructed prior to being used.

© Copyright IBM Corp. 1992, 1996

Some basic guidelines include the following:

- Use a static pointer to the resource rather than a static object.
- Always access the resource using a static function rather than accessing it directly with the static pointer.
- When the resource is accessed through the static function, allocate it if the static pointer is 0.
- Provide a new class that represents the static pointers for a particular class or component. This new class does not require a constructor, but it does require a destructor that destroys the static objects used by the component.

W32s Resource locks are not supported. Instances of the IPrivateResource class can be created and the member functions can be called, but no system resource locks are obtained.

Public Functions

Constructors

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

IPrivateResource

Provides the default constructor. The only way to construct objects of this class is with this, the default constructor.

```
IPrivateResource();
```

Win	PM	Motif
Y	Y	Y

W32s Resource locks are not supported. No resources are created when calling this constructor.

Exceptions	
IAccessError	The private resource object was not created. The memory may be exhausted or the semaphore handle limit on your platform may be exceeded.

~IPrivateResource

```
virtual
  ~IPrivateResource();
```

Win	PM	Motif
Y	Y	Y

W32s Resource locks are not supported. No resources are released when calling this destructor.

Exceptions	
IAccessError	The private resource object was not deleted. The semaphore handle may be invalid.

Inherited Public Functions

IResource		
lock	unlock	

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Resource Handle

Use these members to obtain a handle to the private resource.

handle Returns the handle for the private resource.

```
ISemaphoreHandle&
  handle();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IResource		
handle		

Inherited Protected Data

© IBM Corp. 1992, 1996

IPrivateResource

IBase		
recoverable	**unrecoverable**	

IProcedureAddress

Derivation
IBase
IProcedureAddress

Inherited By None.

Header File iprocadr.hpp

Members

Member	Page	Member	Page
Constructor	724	operator PtrToFnType	725
is32Bit	725	~IProcedureAddress	725

The IProcedureAddress class is a wrapper for dynamic link library (DLL) entry points. This User Interface Class Library uses an operating system function to do a runtime link to the DLL, and it uses another operating system function to acquire the runtime address of a specified function in the DLL. When the link is established and the address is acquired, you can call code that was not accessible to the linker when the executable module was built.

This class provides for the following:

- Constructing objects from entry point names (or ordinal entry point numbers) and DLL objects (or DLL names)
- Calling the entry point without having to cast the generic function type supported by the operating system function that is used to acquire the runtime address
- Managing, along with DynamicLinkLibrary (p. 261), the loading and freeing of the associated DLL module

PM **Customization (Template Argument)**

IProcedureAddress is a template class. Construct objects of this class with the following template argument:

PtrToFnType
Type of the entry point to load or call; this must be a pointer-to-function type.

Motif AIX does not support this class or dynamic link libraries.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

You must provide the following two pieces of information to construct objects of this template class:

- The entry point. You can specify either an entry point name (const char *) or entry point ordinal (unsigned).
- The dynamic link library (DLL) from which to load the entry point. You can specify the DLL either as a reference to an existing IDynamicLinkLibrary (p. 261) object or by the DLL name.

You can also destruct objects of this class.

IProcedureAddress

1
```
IProcedureAddress( unsigned long ordinal,
                   IDynamicLinkLibrary& aDLL );
```

Win	PM	Motif
Y	*Y*	*N*

ordinal Unsigned long value that represents the entry point within the DLL.
aDLL Reference to a dynamic link library object that wrappers the DLL.

Create an object to dynamically load an entry point using the specified entry point ordinal and a reference to an existing IDynamicLinkLibrary object. If you know the ordinal of the entry point and have already loaded the dynamic link library, use this constructor.

2
```
IProcedureAddress( const char* entryPoint,
                   const char* dllName );
```

Win	PM	Motif
Y	*Y*	*N*

entryPoint Pointer that identifies the name of the entry point within the DLL.
dllName Pointer that identifies the DLL name.

Create an object to dynamically load an entry point using the specified entry point name and DLL name. If you know the name of the entry point and have not already loaded the DLL, use this constructor.

3
```
IProcedureAddress( const char* entryPoint,
                   IDynamicLinkLibrary& aDLL );
```

Win	PM	Motif
Y	*Y*	*N*

entryPoint Pointer that identifies the name of the entry point within the DLL.

aDLL Reference to a dynamic link library object that wrappers the DLL.

Create an object to dynamically load an entry point using the specified entry point name and a reference to an existing IDynamicLinkLibrary object. If you know the name of the entry point and have already loaded the DLL, use this constructor.

```
4  IProcedureAddress( unsigned long ordinal,
                      const char* dllName );
```

Win	PM	Motif
Y	*Y*	*N*

ordinal Unsigned long value that represents the entry point within the DLL.
dllName Pointer that identifies the DLL name.

Create an object to dynamically load an entry point using the specified entry point ordinal and DLL name. If you know the ordinal of the entry point and have not already loaded the DLL, use this constructor.

~IProcedureAddress

This destructor frees the IDynamicLinkLibrary object that this class creates during construction. If you need to use the dynamic link library after the IDynamicLinkLibrary object is freed, construct an object of this class using an explicitly created IDynamicLinkLibrary object instead of the dynamic link library name.

```
~IProcedureAddress();
```

Win	PM	Motif
Y	*Y*	*N*

Memory Model

These members help you determine if you are using a 32-bit or a 16-bit memory model.

is32Bit

Determines the memory model that the entry point resides within. If the entry point is 32-bit, this function returns true, and if the entry point is 16-bit, it returns false.

```
Boolean
  is32Bit() const;
```

Win	PM	Motif
Y	*Y*	*N*

Pointers to Functions

These operators permit objects of this class to be used as regular function names.

operator PtrToFnType

Permits objects of this template class to be used in the same way as pointers to functions, which are like regular function names.

© IBM Corp. 1992, 1996

IProcedureAddress

```
operator PtrToFnType() const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IProcessId

Derivation

IBase
  IHandle
    IProcessId

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	727
operator Value	727

The IProcessId class accesses numeric identifiers for processes.

PM IProcessId is an alias for the OS/2 Programmer's Toolkit type PID.

Motif IProcessId is an alias for the system type pid_t.

Public Functions

Constructors

You can construct objects of this class.

IProcessId Constructs objects of this class from a process ID (a value of type IHandle::Value), which defaults to 0.

```
IProcessId( Value pid = 0 );
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

© Copyright IBM Corp. 1992, 1996

IProcessId

```
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**	

IProfile

Derivation

IBase
  IVBase
    IProfile

Inherited By None.

Header File iprofile.hpp

Members

Member	Page	Member	Page
Constructor	730	handle	732
addOrReplaceElementWithKey	733	integerWithKey	735
applicationOrKeyAt	731	name	732
containsApplication	736	numberOfApplications	732
containsKeyName	736	numberOfKeys	732
defaultApplicationName	731	operator =	730
deleteElementWithApplication	734	setDefaultApplicationName	731
deleteElementWithKey	734	systemProfile	736
deleteProfile	734	userProfile	736
elementWithKey	735	~IProfile	731

The IProfile class represents profile data sets. This class provides functions to query and set persistent application data based on application-defined keys.

The profile data set stores information using the following two keys:

- Application name, which is stored as a string
- Key name, which is stored as either a string or an integer

Profile data can be stored in text, binary, or integer format.

PM The objects representing profile data sets are initialization (.ini) files.

Note: The number of .ini files that OS/2 Presentation Manager lets you open at one time varies depending on system limitations.

Win Profile data is stored in the Windows registry instead of an initialization file. The registry preserves case information but ignores the case in all operations. Thus, application names and key names are not case sensitive.

Motif The objects representing profile data sets are Xrm database files.

The profile data set stores information using the following two keys:

© Copyright IBM Corp. 1992, 1996

- Application name, which is stored as a string
- Key name, which is stored as a string

Both of these names must be stored as strings with no embedded spaces. You cannot store key names as integers. You can store the profile data itself as text and integer values using the keys.

Public Functions

Constructors

Use these members to construct, copy, assign, and destruct objects of this class. You can construct objects of this class by using the name of the profile data set or by using the copy constructor to copy one profile to another.

IProfile

1 `IProfile( const char* profileName );`

Win	PM	Motif
Y	Y	Y

profileName
Pointer to the profile name.

Create a profile object with the specified profile name.

Exceptions	
IInvalidParameter	The IProfile object was not constructed. The profile name is missing.
IAccessError	The IProfile object was not constructed. The profile name may be invalid.

2 `IProfile( const IProfile& aProfile );`

Win	PM	Motif
Y	Y	Y

aProfile Reference to an existing profile object.

Create a profile object using a reference to an existing profile object. Use this constructor if you want to make a copy of an existing profile object.

Exceptions	
IAccessError	The IProfile object was not constructed. The profile name may be invalid.

operator = Assigns the member data of an object of this class to another object of this class.

```
IProfile&
  operator =( const IProfile& aProfile );
```

Win	PM	Motif
Y	Y	Y

aProfile Reference to an existing profile object.

Exceptions	
IAccessError	The IProfile object was not assigned. The profile name may be invalid.

~IProfile

This destructor closes the profile data set.

```
virtual
  ~IProfile();
```

Win	PM	Motif
Y	Y	Y

Default Application Name

Use these members to get or set the *default application name*. The default application name is a component of the keys that are used to access the profile. This name is used by default for all reads and writes to the profile.

defaultApplicationName

Returns the default application name.

```
virtual IString
  defaultApplicationName() const;
```

Win	PM	Motif
Y	Y	Y

setDefaultApplicationName

Sets the default application name. All subsequent calls that do not supply an application name to functions of this class use this default name.

```
virtual IProfile&
  setDefaultApplicationName( const char* applName );
```

Win	PM	Motif
Y	Y	Y

applName Pointer to an application name.

Profile Information

Use these members to access general profile information.

applicationOrKeyAt

Returns the application name or key at the specified cursor.

© IBM Corp. 1992, 1996

```
virtual IString
  applicationOrKeyAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

cursor Reference to a nested cursor that has been defined for this class.

Exceptions	
IInvalidParameter	The application name or key was not returned. The cursor is invalid.

handle

Returns the profile handle.

```
virtual IProfileHandle
  handle() const;
```

Win	PM	Motif
N	Y	Y

name

Returns the profile's name.

```
virtual IString
  name() const;
```

Win	PM	Motif
Y	Y	Y

numberOfApplications

Returns the number of application names in the profile.

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

Win	PM	Motif
Y	Y	I

numberOfKeys

Returns the number of keys for the application name.

```
virtual unsigned long
  numberOfKeys( const char* applName = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Exceptions	
IInvalidParameter	The number of keys was not returned. The application name is missing and a default has not been specified.

Reading and Writing Data

Use these members to read or write application data for an application.

addOrReplaceElementWithKey

Writes the specified numeric, text, or binary data. You must specify text and binary data as an IString (Vol. I).

1

```
virtual IProfile&
  addOrReplaceElementWithKey( const char* key,
                              const IString& data,
                              const char* applName = 0 );
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.

data Reference to an IString that contains text or binary data to use for a write.

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Use this version of the function if your data is in a text or binary format.

Win Storing data larger than two kilobytes is not recommended due to performance and size restrictions. For data larger than two kilobytes, applications should store the data as a file and store the file name in the profile.

Motif AIX does not support the writing of binary data.

Exceptions	
IInvalidParameter	The application data was not written. The key name is missing.
IInvalidParameter	The application data was not written. The application name is missing and a default has not been specified.
IAccessError	The application data was not written. The application name or key may be invalid.

2

```
virtual IProfile&
  addOrReplaceElementWithKey( const char* key,
                              long data,
                              const char* applName = 0 );
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.
data Long integer numeric to use for a write.
applName Pointer to an application name. If you do not supply a name, the default application name is used.

Use this version of the function if your data is numeric.

Motif AIX does not support the writing of binary data.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidParameter	The application data was not written. The key name is missing.
IInvalidParameter	The application data was not written. The application name is missing and a default has not been specified.
IAccessError	The application data was not written. The application name or key may be invalid.

deleteElementWithApplication

Removes the data for all keys for the specified application name.

```
virtual IProfile&
  deleteElementWithApplication( const char* applName = 0 );
```

Win	PM	Motif
Y	Y	Y

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Exceptions	
IInvalidParameter	The application data was not removed. The application name is missing and a default has not been specified.
IAccessError	The application data was not removed. The application name may be invalid.

deleteElementWithKey

Removes the data at the specified application and key name pair.

```
virtual IProfile&
  deleteElementWithKey( const char* key,
                        const char* applName = 0 );
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Exceptions	
IInvalidParameter	The application data was not removed. The key name is missing.
IInvalidParameter	The application data was not removed. The application name is missing and a default has not been specified.
IAccessError	The application data was not removed. The application name or key may be invalid.

deleteProfile

Removes the specified profile from the system.

```
static void
  deleteProfile( const char* profileName );
```

Win	PM	Motif
Y	Y	N

elementWithKey

Reads the application data and returns it as an IString (Vol. I).

```
virtual IString
  elementWithKey( const char* key,
                  const char* applName = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Exceptions	
IInvalidParameter	The application data was not read. The key name is missing.
IInvalidParameter	The application data was not read. The application name is missing and a default has not been specified.
IAccessError	The application data was not read. The application name or key may be invalid.
IInvalidRequest	The application data was not read. The application name or key may be invalid.

integerWithKey

Reads the application data and returns it as a long integer.

```
virtual long
  integerWithKey( const char* key,
                  const char* applName = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Exceptions	
IInvalidParameter	The application data was not read. The key name is missing.
IInvalidParameter	The application data was not read. The application name is missing and a default has not been specified.
IAccessError	The application data was not read. The application name or key may be invalid.

Special Profiles

Use these members to return the special objects of this class that correspond to the system and user profiles.

© IBM Corp. 1992, 1996

IProfile

systemProfile

Returns the system profile.

```
static IProfile
  systemProfile();
```

Win	PM	Motif
Y	Y	I

userProfile

Returns the user profile.

```
static IProfile
  userProfile();
```

Win	PM	Motif
Y	Y	I

Testing

Use these members to test a profile for an application name or an application and key name pair.

containsApplication

If the profile contains data for the specified application name, true is returned.

```
virtual Boolean
  containsApplication( const char* applName ) const;
```

Win	PM	Motif
Y	Y	Y

applName Pointer to an application name.

Exceptions	
IInvalidParameter	The profile query was not returned. The application name is missing.

containsKeyName

If the profile contains data for the specified application and key pair, true is returned.

```
virtual Boolean
  containsKeyName( const char* key,
                   const char* applName = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

key Pointer to a key name.

applName Pointer to an application name. If you do not supply a name, the default application name is used.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

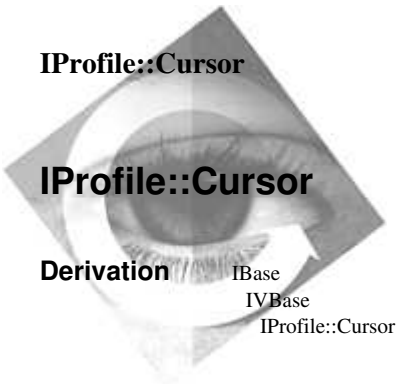
IBase		
recoverable	**unrecoverable**	

Nested Classes

IProfile contains the following nested classes:

IProfile::Cursor (see page 738)

© IBM Corp. 1992, 1996



IProfile::Cursor

Derivation

IBase
IVBase
IProfile::Cursor

Inherited By None.

Header File iprofile.hpp

Members

Member	Page	Member	Page
Constructor	738	setToLast	740
Cursor	738	setToNext	740
invalidate	739	setToPrevious	740
isValid	739	~Cursor	739
setToFirst	739		

The IProfile::Cursor class creates and manages the cursor for an IProfile (p. 729) object. IProfile uses this cursor to iterate through the application names or keys in a profile. In the same way that you can use a cursor to iterate through the objects in a collection, you can use this cursor to iterate through a profile one item at a time.

Public Functions

Constructors

Use these members to construct and destruct objects of this nested class. You can construct objects of this class by using a reference to the profile, or a reference to the profile data and an application name.

Cursor

Note: The cursor object is invalidated for the current state of application names or key names in the profile data set.

1 `Cursor( IProfile& profile );`

Win	PM	Motif
Y	Y	Y

profile Reference to the profile object.

You can construct an object of this class by using a reference to the profile. Use this constructor to create a cursor to iterate through the application names.

© Copyright IBM Corp. 1992, 1996

2

```
Cursor( IProfile& profile,
        const char* applName );
```

Win	PM	Motif
Y	Y	Y

profile Reference to the profile object.

applName Pointer to an application name.

You can construct an object of this class by using a reference to the profile and an application name. Use this constructor to create a cursor to iterate through the keys.

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Profile Iteration

Use these members to iterate through the application names or keys in a profile.

invalidate

Marks the cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid

Determines if the cursor is still valid. If the cursor is valid, true is returned. If the parameter, *checkFile*, is set to true, this function checks to see if the data set's application and key information has been changed by another application while your application still has the data set open.

```
virtual Boolean
  isValid( Boolean checkFile = false ) const;
```

Win	PM	Motif
Y	Y	Y

checkFile Boolean flag to query if another application has changed the application and key information, thus rendering the cursor invalid.

PM If *checkFile* is set to true, this function determines whether the file application and key information have been changed by another application while your application still has them open.

Motif AIX does not support *checkFile*.

setToFirst

Sets the cursor's position to the first application or key.

© IBM Corp. 1992, 1996

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToLast Sets the cursor's position to the last application or key.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Sets the cursor's position to the next application or key.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Set the cursor's position to the previous application or key.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IProfileHandle

Derivation

IBase
  IHandle
    IProfileHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	741
operator _XrmHashBucketRec *	742

The IProfileHandle class accesses profiles.

PM IProfileHandle is an alias for the OS/2 Programmer's Toolkit type HINI.

Motif IProfileHandle is an alias for the X-Motif types XrmHashBucket and XrmDatabase.

Public Functions

Constructors

You can construct objects of this class.

IProfileHandle

1 `IProfileHandle( Value hini = 0 );`

Win	PM	Motif
Y	*Y*	*Y*

You can construct objects of this class from a profile handle (a value of type IHandle::Value), which defaults to 0.

2 `IProfileHandle( _XrmHashBucketRec* hini );`

Win	PM	Motif
N	*N*	*Y*

You can construct objects of this class from a profile handle.

© Copyright IBM Corp. 1992, 1996

IProfileHandle

```
IProfileHandle( int hini );
```

Win	PM	Motif
N	N	Y

You can construct objects of this class from a profile handle.

Operators

These members are the operators for this class.

operator _XrmHashBucketRec *

Returns the handle value as a native X-Toolkit handle object.

```
operator _XrmHashBucketRec *() 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**	

IProgressIndicator

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IProgressIndicator

Inherited By

ISlider

Header File

islider.hpp

Members

Member	Page	Member	Page
Constructor	748	isPMCompatible	756
alignBottom	764	isRibbonStripEnabled	753
alignCentered	764	isSnapToTickEnabled	755
alignLeft	764	isVertical	745
alignment	745	moveArmToPixel	746
alignRight	764	moveArmToTick	746
alignTop	765	moveSizeTo	761
armChangeId	764	numberOfTicks	757
armPixelOffset	746	pmCompatible	766
armRange	746	primaryScale	753
armTickOffset	746	primaryScale1	766
backgroundColor	747	primaryScale2	766
border3D	765	registerCallbacks	762
calcMinimumSize	763	ribbonStrip	767
classDefaultStyle	765	scaleId	764
convertToGUIStyle	756	setBackgroundColor	747
defaultStyle	756	setDefaultStyle	756
disableDrawItem	751	setForegroundColor	748
disableRibbonStrip	753	setHomePosition	752
disableSnapToTick	755	setPrimaryScale	753
enableDrawItem	751	setShaftBreadth	754
enableNotification	752	setShaftPosition	754
enableRibbonStrip	753	setTickLength	757
enableSnapToTick	755	setTicks	758
foregroundColor	747	setTickText	759
handleDrawItem	765	shaftPosition	754
homeBottom	765	shaftSize	755
homeLeft	765	snapToTickMark	767
homePosition	752	tickLength	759
homeRight	766	tickPosition	760
homeTop	766	tickSpacing	760
horizontal	766	tickText	760
initialize	762	unregisterCallbacks	762
isDrawItemEnabled	751	vertical	767

© Copyright IBM Corp. 1992, 1996

IProgressIndicator

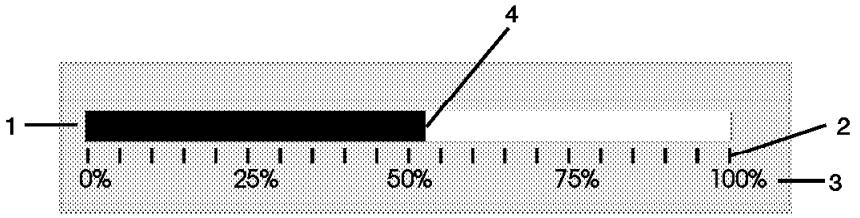
Member	Page
~IProgressIndicator	750

The IProgressIndicator class is a read-only version of the slider control. ISlider (p. 874) describes the slider control.

Handlers derived from the following classes handle events for IProgressIndicator objects:

- IFocusHandler (p. 322)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISliderDrawHandler (p. 892)

Typically, a progress indicator with the following components displays the percentage of a task completed by filling in the indicator shaft as the task progresses.



1 - Slider shaft A track for the slider arm to move along.

2 - Tick mark A mark that indicates an incremental value in the slider's scale.

3 - Tick text A label that indicates the value of the tick mark.

4 - Slider arm The slider arm shows the level of progress by its position on the slider shaft. The user cannot move the arm of the progress indicator. The arm is barely visible. You can only change the slider arm's position through your code.

If the ribbonStrip style is enabled, the progress indicator's shaft fills with color as the arm is moved.

By default, the User Interface Class Library creates a horizontal progress indicator and centers it in the window with its ticks and text above it. The arm starts from the left edge and the shaft fills with color as the arm moves to the right. You can also construct a progress indicator with the following characteristics:

- The ticks and text are below the shaft.
- The indicator is at various positions in the window.
- The arm starts from the right edge.

- The indicator is vertical.

Win Native progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not display a visible slider arm. You must use the ribbonStrip style to indicate the arm position for native progress indicators.

Motif The IProgressIndicator constructor creates objects of this class using the following Motif widgets:

- An XmDrawingArea widget is created with an XmScrollBar child if the ribbonStrip style is enabled; otherwise, it is created with an XmScale child. IWindow::handle returns the handle of the XmDrawingArea widget. The XmDrawingArea widget also has one or two XmForm widget children.
- The XmForm widgets contain the tick marks and tick text.
- The tick marks are XmSeparator widgets, and the tick text is implemented with XmLabel widgets.
- XmSeparator widgets are created for each widget during the construction of the IProgressIndicator or during processing of the setTicks member function.
- The widgets representing ticks are managed when their size is set to a nonzero value.
- XmLabel widgets are created for the tick text only when setTickText is called.

Because IProgressIndicator objects are read-only, XtUninstallTranslations is used to remove the translations from the underlying Scrollbar widget.

Public Functions

Alignment

Use these members to query the alignment of the progress indicator object. The alignment can only be set in the constructor of the progress indicator object.

alignment Returns the current alignment of this progress indicator object. The returned value is an enumerator provided by Alignment (p. 768).

```
Alignment
  alignment() const;
```

Win	PM	Motif
Y	*Y*	*Y*

isVertical If the progress indicator is oriented vertically, true is returned. If it is oriented horizontally, false is returned.

© IBM Corp. 1992, 1996

```
Boolean
  isVertical() const;
```

Win	PM	Motif
Y	Y	Y

Arm Operations

Use these members to set and query attributes of the progress indicator arm.

armPixelOffset

Returns the offset, in pixels, of the arm from the home position.

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

Win	PM	Motif
Y	Y	Y

armRange

Returns the number of pixels over which the arm can move.

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

Win	PM	Motif
Y	Y	Y

armTickOffset

Returns the position of the arm as a tick number. Ticks are numbered starting at 0.

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

Win	PM	Motif
Y	Y	Y

moveArmToPixel

Moves the arm to a pixel offset relative to the home position.

armOffset Number of pixels to offset the arm.

```
virtual IProgressIndicator&
  moveArmToPixel( unsigned long armOffset );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The arm offset is invalid.

moveArmToTick

Moves the arm to a specified tick relative to the home position. Ticks are numbered starting at 0.

If the progress indicator window has no size and you specified tick spacing on the constructor, using this function can cause an exception.

tickNumber
Number of ticks to offset the arm.

```
virtual IProgressIndicator&
  moveArmToTick( unsigned long tickNumber );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The tick index is invalid.

Colors

Use these members to query, set, and reset the colors of the progress indicator.

backgroundColor

Returns the background color value of the progress indicator area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	Y	Y

This member is overridden in this derived class for specific operating system behavior.

foregroundColor

Returns the foreground color value of the progress indicator area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
N	N	Y

This member is overridden in this derived class for specific operating system behavior.

setBackgroundColor

Sets the background color to the specified color.

```
virtual IProgressIndicator&
  setBackgroundColor( const IColor& color );
```

Win	PM	Motif
N	N	Y

This member is overridden in this derived class for specific operating system behavior.

© IBM Corp. 1992, 1996

IProgressIndicator

setForegroundColor

Sets the color for the ticks and label text to the specified color.

```
virtual IProgressIndicator&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
N	*N*	*Y*

Win Sets only the color for label text for native Windows progress indicators (that is, progress indicators constructed without the pmCompatible style).

This member is overridden in this derived class for specific operating system behavior.

Constructors

You can construct and destruct objects of this class.

In addition, the User Interface Class Library provides a protected constructor used by the ISlider (p. 874) class during construction.

IProgressIndicator

1
```
IProgressIndicator( const IWindowHandle& handle );
```

Win	PM	Motif
Y	*Y*	*N*

You can construct objects of this class by specifying a window handle.

handle The window handle of the progress indicator control.

Win The constructors wrapping an existing progress indicator are not supported for native Windows progress indicators.

Motif The constructors wrapping an existing progress indicator are not available in the AIX environment.

2
```
IProgressIndicator( unsigned long windowId,
                    IWindow* parent,
                    IWindow* owner,
                    const IRectangle& initial,
                    unsigned long numberOfTicks,
                    unsigned long tickSpacing = 0,
                    const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	*Y*	*Y*

You can construct objects of this class by using this primary constructor.

windowId A unique ID for the progress indicator control.

parent The parent window.

owner The owner window.

initial A rectangle for the progress indicator control. It specifies the initial position and size of the progress indicator you are constructing. The default is the rectangle constructed by the default IRectangle (Vol. I) constructor.

numberOfTicks
The number of ticks to place on the primary scale of the progress indicator.

tickSpacing
The number of pixels between ticks. The default is 0. The default causes the progress indicator to evenly space the ticks on the shaft. Otherwise, the length of the slider shaft is based on the *tickSpacing* and the *numberOfTicks* you specify. Optional.

The tick spacing can also be changed dynamically by using the setTicks member function.

style The style of the control. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

3 `IProgressIndicator( unsigned long windowId,`

Win	PM	Motif
Y	*Y*	*Y*

```
                       IWindow* parent,
                       IWindow* owner,
                       const IRectangle& initial,
                       unsigned long scale1NumberOfTicks,
                       unsigned long scale1TickSpacing,
                       unsigned long scale2NumberOfTicks,
                       unsigned long scale2TickSpacing = 0,
                       const Style& style = defaultStyle ( ) );
```

You can construct objects of this class by specifying the number of ticks and spacing for both scale 1 and scale 2.

Note: Various repaint problems can occur when the progress indicator control is resized using IWindow::sizeTo (p. 1079) or IWindow::moveSizeTo (p. 1078). The problems occur when the width of the window is less than the width of the entire progress indicator. This might happen when you attempt to show the control with a nonzero rectangle that is too small for the tick spacing you specified. If you use the default tick spacing, the progress indicator is able to resize itself for the new rectangle's size.

© IBM Corp. 1992, 1996

IProgressIndicator

windowId A unique ID for the progress indicator control.

parent The parent window.

owner The owner window.

initial A rectangle for the progress indicator control. It specifies the initial position and size of the progress indicator you are constructing. The default is the rectangle constructed by the default IRectangle (Vol. I) constructor.

scale1NumberOfTicks
The number of ticks to place on scale 1 of the progress indicator.

scale1TickSpacing
The number of pixels between ticks on scale 1.

scale2NumberOfTicks
The number of ticks to place on scale 2 of the progress indicator.

scale2TickSpacing
The number of pixels between ticks on scale 2. Optional.

style The style of the control. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

4
```
IProgressIndicator( unsigned long windowId,
                    IWindow* parent );
```

Win	PM	Motif
Y	Y	N

You can construct objects of this class by specifying a parent window and the ID of a progress indicator control.

windowId A unique ID for the progress indicator control.
parent The parent window.

Win The constructors wrapping an existing progress indicator are not supported for native Windows progress indicators.

~IProgressIndicator

```
virtual
  ~IProgressIndicator();
```

Win	PM	Motif
Y	Y	Y

Drawing

Use these members to query and change the handleDrawItem style of the progress indicator object. The handleDrawItem style specifies that events will be created during progress indicator drawing. You can process these events to implement your own drawing by deriving a handler from ISliderDrawHandler (p. 892) and attaching your handler to the progress indicator object.

disableDrawItem

Sets the handleDrawItem (p. 765) style to off.

```
virtual IProgressIndicator&
  disableDrawItem();
```

Win	PM	Motif
Y	*Y*	*I*

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support the handleDrawItem style. Therefore, this member function will have no effect for native progress indicators.

Motif The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

enableDrawItem

Enables or disables the handleDrawItem (p. 765) style for the progress indicator.

```
virtual IProgressIndicator&
  enableDrawItem( Boolean enableDrawItem = true );
```

Win	PM	Motif
Y	*Y*	*I*

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support the handleDrawItem style. Therefore, this member function will have no effect for native progress indicators.

Motif The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

isDrawItemEnabled

If the handleDrawItem (p. 765) style is set, true is returned. Otherwise, false is returned.

```
Boolean
  isDrawItemEnabled() const;
```

Win	PM	Motif
Y	*Y*	*I*

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support the handleDrawItem style. Therefore, this member function will always return false for native progress indicators.

© IBM Corp. 1992, 1996

Motif The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

Home Position

Use these members to query and modify the home position of the progress indicator object. The home position specifies from which side the progress indicator's tick marks are indexed. The home position is also used to indicate from which side of the progress indicator a ribbon strip, if specified, is attached.

homePosition

Returns the current home position for this progress indicator object. The returned value is an enumerator provided by HomePosition (p. 768).

```
HomePosition
  homePosition() const;
```

Win	PM	Motif
Y	Y	Y

setHomePosition

Sets the home position of the progress indicator.

home Enumerator value for the home position. Use the enumeration HomePosition (p. 768) to specify the home position.

```
virtual IProgressIndicator&
  setHomePosition( HomePosition home = homeBottomLeft );
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify notifications sent to observer objects.

enableNotification

Enables the progress indicator to send notifications to any observer objects.

```
virtual IProgressIndicator&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Primary Scale

Use these members to query and modify the primary scale of the progress indicator object. The primary scale specifies from which scale the progress indicator is positioned.

primaryScale Returns the current primary scale of this progress indicator object. The returned value is an enumerator provided by Scale (p. 768).

```
Scale
  primaryScale() const;
```

Win	PM	Motif
Y	Y	Y

setPrimaryScale

Sets the primary scale of the progress indicator.

primaryScale

Use the enumeration Scale (p. 768) to specify which scale to set.

```
virtual IProgressIndicator&
  setPrimaryScale( Scale primaryScale = scale1 );
```

Win	PM	Motif
Y	Y	Y

Ribbon Strip

Use these members to query and change the ribbonStrip style of the progress indicator object. The ribbonStrip style specifies that the progress indicator's shaft fills with color between the arm and home position when the arm is moved.

disableRibbonStrip

Sets the ribbonStrip (p. 767) style to off.

```
virtual IProgressIndicator&
  disableRibbonStrip();
```

Win	PM	Motif
Y	Y	Y

enableRibbonStrip

Enables or disables the ribbonStrip (p. 767) style for the progress indicator.

```
virtual IProgressIndicator&
  enableRibbonStrip( Boolean enableRibbonStrip = true );
```

Win	PM	Motif
Y	Y	Y

isRibbonStripEnabled

If the ribbonStrip (p. 767) style is set, true is returned. Otherwise, false is returned.

```
Boolean
  isRibbonStripEnabled() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Shaft Operations

Use these members to set and query attributes of the progress indicator's shaft.

setShaftBreadth

Sets the shaft width in pixels for vertical progress indicators. Sets the shaft height in pixels for horizontal progress indicators.

breadth Number of pixels.

```
virtual IProgressIndicator&
  setShaftBreadth( unsigned long breadth );
```

Win	PM	Motif
Y	Y	Y

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support shaft sizing. Therefore, this member function has no effect for native Windows progress indicators.

Exceptions	
IInvalidParameter	The breadth is invalid.

setShaftPosition

Sets the position, in pixels, of the origin point of the shaft. The position of the shaft is relative to the origin of the progress indicator window.

```
virtual IProgressIndicator&
  setShaftPosition( const IPoint& position );
```

Win	PM	Motif
Y	Y	I

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support shaft positioning. Therefore, this member function has no effect for native Windows progress indicators.

Motif The AIX release ignores this function.

Exceptions	
IInvalidParameter	The position is invalid.

shaftPosition

Returns the origin point of the shaft relative to the origin corner of the progress indicator window.

```
virtual IPoint
  shaftPosition() const;
```

Win	PM	Motif
Y	Y	Y

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support shaft positioning. Therefore, this

member function returns a default IPoint object for native Windows progress indicators.

shaftSize Returns the size of the shaft in pixels.

```
virtual ISize
  shaftSize() const;
```

Win	PM	Motif
Y	Y	Y

Win Native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support shaft sizing. Therefore, this member function returns a default ISize object for native Windows progress indicators.

Snap Support

Use these members to query and change the snapToTickMark style of the progress indicator object. The snapToTickMark style specifies that the progress indicator's arm snaps to the nearest tick when moved between two ticks.

disableSnapToTick

Sets the snapToTickMark (p. 767) style to off.

```
virtual IProgressIndicator&
  disableSnapToTick();
```

Win	PM	Motif
Y	Y	Y

enableSnapToTick

Enables or disables the snapToTickMark (p. 767) style for the progress indicator.

```
virtual IProgressIndicator&
  enableSnapToTick( Boolean enableSnapToTick = true );
```

Win	PM	Motif
Y	Y	Y

isSnapToTickEnabled

If the snapToTickMark (p. 767) style is set, true is returned. Otherwise, false is returned.

```
Boolean
  isSnapToTickEnabled() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the progress indicator styles. You can use these styles with the styles in the following classes:

© IBM Corp. 1992, 1996

IProgressIndicator

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 765) unless you have changed the style using setDefaultStyle (p. 756).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

isPMCompatible

Use this member to query whether the progress indicator was constructed with the pmCompatible (p. 766) style.

```
Boolean
  isPMCompatible() const;
```

Win	PM	Motif
Y	Y	I

setDefaultStyle

Sets the default style for all subsequent progress indicators.

style Use the styles provided by IProgressIndicator Styles (p. 763) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Tick Operations

Use these members to set and query attributes of tick marks.

numberOfTicks

Returns the number of ticks for the specified scale.

```
unsigned long
  numberOfTicks( Scale scale ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to query the progress indicator's control data has failed.

setTickLength

Sets the length of one or all ticks on the progress indicator scale.

Note: Because ticks are created with 0 length, they are not visible initially.

tickNumber
A 0-based tick number that is relative to the home position.

length Length, in pixels, of the tick.

1

```
virtual IProgressIndicator&
  setTickLength( unsigned long length );
```

Win	PM	Motif
Y	Y	Y

Win The first and last ticks in a native Windows progress indicator (that is, a progress indicator constructed without the pmCompatible (p. 766) style) are always visible. Therefore, this member function has no effect for the first and last ticks. All other ticks are made visible by setting a non-zero tick length and they are removed by setting the tick length to zero.

Motif The XmSeparator widget used for a tick mark is not managed until it has a nonzero length. Due to a Motif restriction, you cannot use a length of 1. If you specify a length of 1, this function uses a length of 2.

Exceptions	
IInvalidParameter	The tick length is invalid.

2

```
virtual IProgressIndicator&
  setTickLength( unsigned long tickNumber,
                 unsigned long length );
```

Win	PM	Motif
Y	Y	Y

Win The first and last ticks in a native Windows progress indicator (that is, a progress indicator constructed without the pmCompatible (p. 766) style) are always visible. Therefore, this member function has no effect for the first and last ticks. All other ticks are made visible by setting a non-zero tick length for the tick, and they are removed by setting the tick length to zero.

© IBM Corp. 1992, 1996

Motif The XmSeparator widget used for a tick mark is not managed until it has a nonzero length. Due to a Motif restriction, you cannot use a length of 1. If you specify a length of 1, this function uses a length of 2.

Exceptions	
IInvalidParameter	Either the tick index or the tick length is invalid.

setTicks

Sets the following for one (or both) of the scales:

- Number of ticks
- Number of pixels between ticks

1

```
virtual IProgressIndicator&
  setTicks( Scale scale,
            unsigned long numberOfTicks,
            unsigned long tickSpacing = 0 );
```

Win	PM	Motif
Y	Y	Y

Sets the scale information for either scale 1 or scale 2.

scale Identifier of which scale should be set to the values that follow. This value is one of the predefined Scale enum values.

numberOfTicks
The number of ticks to place on the scale of the progress indicator.

tickSpacing
The number of pixels between ticks on the scale.

Exceptions	
IAccessError	The operating system's request to set the progress indicator's control data has failed.

2

```
virtual IProgressIndicator&
  setTicks( unsigned long scale1NumberOfTicks,
            unsigned long scale2NumberOfTicks,
            unsigned long scale1TickSpacing = 0,
            unsigned long scale2TickSpacing = 0 );
```

Win	PM	Motif
Y	Y	Y

Sets the scale information for both scale1 and scale2.

scale1NumberOfTicks
The number of ticks to place on scale 1 of the progress indicator.

scale2NumberOfTicks
The number of ticks to place on scale 2 of the progress indicator.

scale1TickSpacing
The number of pixels between ticks on scale 1.

scale2TickSpacing
The number of pixels between ticks on scale 2.

Exceptions	
IAccessError	The operating system's request to set the progress indicator's control data has failed.

setTickText

Sets the text associated with the tick at the specified index. A tick need not be visible to have text associated with it.

tickNumber
A 0-based tick number relative to the home position.

text The text you want to label the tick with.

1

```
virtual IProgressIndicator&
  setTickText( unsigned long tickNumber,
               const char* text );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	Either the tick index or the tick text is invalid.

2

```
virtual IProgressIndicator&
  setTickText( unsigned long tickNumber,
               const IResourceId& text );
```

Win	PM	Motif
Y	Y	Y

tickLength

Returns the length, in pixels, of the tick at the specified index.

tickNumber
A 0-based tick number relative to the home position.

```
unsigned long
  tickLength( unsigned long tickNumber ) const;
```

Win	PM	Motif
Y	Y	Y

Win The first and last ticks in a native Windows progress indicator (that is, a progress indicator constructed without the pmCompatible (p. 766) style) are always visible. The length returned for these ticks is 1. The tick length for all other ticks is either 1 if the tick is visible or 0 if the tick is not visible.

Exceptions	
IInvalidParameter	The tick index is invalid.

© IBM Corp. 1992, 1996

tickPosition

Returns the pixel position of the tick at the specified index. Ticks are numbered starting with 0. The position returned is the end of the tick mark closest to the slider shaft.

tickNumber
A 0-based tick number relative to the home position.

```
IPoint
  tickPosition( unsigned long tickNumber ) const;
```

Win	PM	Motif
Y	Y	Y

Win This member function is not supported for native Windows progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style). A default IPoint object is returned for native Windows progress indicators.

Exceptions	
IInvalidParameter	The tick index is invalid.

tickSpacing

Returns the number of pixels between ticks for the specified scale.

scale Use the enumeration Scale (p. 768) to specify for which scale the tick spacing is set.

```
unsigned long
  tickSpacing( Scale scale ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to query the progress indicator's control data has failed.

tickText

Returns the text associated with the tick at the specified index. Ticks are numbered starting with 0.

tickNumber
A 0-based tick number relative to the home position.

```
IString
  tickText( unsigned long tickNumber ) const;
```

Win	PM	Motif
Y	Y	Y

Window Positioning

Use these members to set and query the size and position of progress indicators. Unless otherwise noted, the orientation of the coordinates accepted and returned by these members is the

application orientation. For more information about coordinate orientation, see ICoordinateSystem (p. 230).

moveSizeTo Changes the position and size of the progress indicator window using a lower-left origin coordinate system.

rectangle A rectangle for the progress indicator control.

```
virtual IProgressIndicator&
  moveSizeTo( const IRectangle& rectangle );
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

Inherited Public Functions

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 (p. 1044) are not shown.

Protected Functions

Constructors

You can construct and destruct objects of this class.

In addition, the User Interface Class Library provides a protected constructor used by the ISlider (p. 874) class during construction.

© IBM Corp. 1992, 1996

IProgressIndicator

:h14IProgressIndicator

```
IProgressIndicator();
```

Win	PM	Motif
Y	*Y*	*Y*

Used by derived classes to construct objects of this class. This is the default constructor and accepts no parameters.

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Called during construction of the IProgressIndicator window.

This function registers all possible callbacks and X event handlers to this IProgressIndicator (p. 1044) for events it receives. IHandler (p. 411) derived classes later determine which events they will process. If classes you derive from IProgressIndicator override the registerCallbacks member function, the override must call Inherited::registerCallbacks to register the applicable Motif callbacks and X event handlers for a progress indicator control.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes all callbacks and X event handlers from this IWindow (p. 1044) that were added by registerCallbacks (p. 762). If classes you derive from IProgressIndicator override the unregisterCallbacks member function, the override must call Inherited::unregisterCallbacks to unregister the applicable Motif callbacks and X event handlers for a progress indicator control.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Implementation

These members provide utilities used to implement this class.

initialize Creates a progress indicator or slider control. Several of the IProgressIndicator and ISlider constructors use this function.

windowId A unique ID for the progress indicator control.

parent The parent window.
owner The owner window.
style Specifies the progress indicator or slider style.
initial A rectangle defining the size and placement of the control window.
sliderData Specifies the number of ticks and tick spacing for each scale.

```
void
  initialize( unsigned long windowId,
              unsigned long parent,
              unsigned long owner,
              unsigned long style,
              const IRectangle& initial,
              void* sliderData );
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns an ISize (Vol. I) indicating the minimum size of the progress indicator. ICanvas (Vol. III) and its derived classes use this function.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify notifications sent to observer objects.

© IBM Corp. 1992, 1996

armChangeId Notification identifier provided to observers when the arm of an IProgressIndicator window is moved to a new location. IProgressIndicator provides the new arm position in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  armChangeId;
```

Win	PM	Motif
Y	Y	Y

scaleId Notification identifier provided to observers when the scale style of an IProgressIndicator window changes. IProgressIndicator provides the new scale value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is one of the predefined Scale enum values.

```
static INotificationId const
  scaleId;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the progress indicator styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

alignBottom Sets the progress indicator to be at the bottom of the window.

```
static const Style
  alignBottom;
```

Win	PM	Motif
Y	Y	Y

alignCentered

Sets the progress indicator to be at the center of the window.

```
static const Style
  alignCentered;
```

Win	PM	Motif
Y	Y	Y

alignLeft Sets the progress indicator to be at the left side of the window.

```
static const Style
  alignLeft;
```

Win	PM	Motif
Y	Y	Y

alignRight Sets the progress indicator to be at the right side of the window.

```
static const Style
  alignRight;
```

Win	PM	Motif
Y	Y	Y

alignTop Sets the progress indicator to be at the top of the window.

```
static const Style
  alignTop;
```

Win	PM	Motif
Y	Y	Y

border3D Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	I	I

Win This style is ignored on Windows NT 3.51 with Program Manager.

W32s This style is ignored on WIn32s.

classDefaultStyle

Provides the original default style for this class, which is the following: IWindow::visible | IProgressIndicator::horizontal | IProgressIndicator::alignCentered | IProgressIndicator::homeLeft | IProgressIndicator::ribbonStrip | IProgressIndicator::primaryScale1.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

handleDrawItem

An event is dispatched to the control whenever the shaft, ribbon strip, arm, and background are to be drawn.

```
static const Style
  handleDrawItem;
```

Win	PM	Motif
Y	Y	I

Win Native progress indicators (that is, progress indicators constructed without the pmCompatible (p. 766) style) do not support the handleDrawItem style.

Motif AIX does not support the handleDrawItem style.

homeBottom Specifies that the progress indicator's arm and scale start at the bottom.

```
static const Style
  homeBottom;
```

Win	PM	Motif
Y	Y	Y

homeLeft Specifies that the progress indicator's arm and scale start at the left.

© IBM Corp. 1992, 1996

```
static const Style
  homeLeft;
```

Win	PM	Motif
Y	Y	Y

homeRight Specifies that the progress indicator's arm and scale start at the right.

```
static const Style
  homeRight;
```

Win	PM	Motif
Y	Y	Y

homeTop Specifies that the progress indicator's arm and scale start at the top.

```
static const Style
  homeTop;
```

Win	PM	Motif
Y	Y	Y

horizontal Specifies that the progress indicator is built horizontally.

```
static const Style
  horizontal;
```

Win	PM	Motif
Y	Y	Y

pmCompatible

Specifies that the progress indicator is compatible with the PM version of this control in both appearance and behavior.

```
static const Style
  pmCompatible;
```

Win	PM	Motif
Y	Y	Y

primaryScale1

Specifies that scale 1 is used for positioning the arm and is located above or to the right of the progress indicator.

```
static const Style
  primaryScale1;
```

Win	PM	Motif
Y	Y	Y

primaryScale2

Specifies that scale 2 is used for positioning the arm and is located below or to the left of the progress indicator.

```
static const Style
  primaryScale2;
```

Win	PM	Motif
Y	Y	Y

ribbonStrip Specifies that the progress indicator's shaft fills with color between the arm and home position when the arm is moved. This style is used to improve the visual clarity of the current value.

```
static const Style
  ribbonStrip;
```

Win	PM	Motif
Y	*Y*	*Y*

snapToTickMark
Specifies that the progress indicator's arm snaps to the nearest tick when moved between two ticks.

```
static const Style
  snapToTickMark;
```

Win	PM	Motif
Y	*Y*	*Y*

vertical Specifies that the progress indicator is built vertically.

```
static const Style
  vertical;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

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 (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IProgressIndicator contains the following nested classes:

IProgressIndicator::Style (see page 770)

Alignment

```
Alignment {
    topRight,
    bottomLeft,
    centered
    };
```

Use these enumerators to specify where to place the progress indicator:

topRight
Aligns a vertical progress indicator to the right of the control window. A horizontal progress indicator is aligned to the top.

bottomLeft
Aligns a vertical progress indicator to the left of the control window. A horizontal progress indicator is aligned to the bottom.

centered
Centers the progress indicator.

HomePosition

```
HomePosition {
    homeTopRight,
    homeBottomLeft
    };
```

Use these enumerators to specify the home position of the progress indicator:

homeBottomLeft
The home position is at the bottom for a vertical progress indicator and at the left for a horizontal progress indicator.

homeTopRight
The home position is at the top for a vertical progress indicator and at the right for a horizontal progress indicator.

```
Scale {
    scale1,
    scale2
    };
```

Use these enumerators to specify which scale you are referring to:

scale1
Scale 1 is above a horizontal progress indicator or to the right of a vertical progress indicator.

scale2
Scale 2 is below a horizontal progress indicator or to the left of a vertical progress indicator.

© IBM Corp. 1992, 1996

IProgressIndicator::Style

Derivation IBase
IBitFlag
IProgressIndicator::Style

Inherited By None.

Header File islider.hpp

The nested class IProgressIndicator::Style provides a set of valid styles for the IProgressIndicator (p. 743) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IProgressIndicatorNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        IProgressIndicatorNotifyHandler

Inherited By None.

Header File islidenh.hpp

Members

Member	Page
Constructor	771
dispatchHandlerEvent	772
~IProgressIndicatorNotifyHandler	771

Instances of the IProgressIndicatorNotifyHandler class process events for all classes of progress indicators or sliders.

This class is designed to handle events that require the class to generate a notification. If notifications are enabled for this class, a notification will be generated and sent to all observers when the proper conditions for the specific notification exist.

Public Functions

Constructors

Use these functions to construct and destruct objects of this class.

IProgressIndicatorNotifyHandler

```
IProgressIndicatorNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IProgressIndicatorNotifyHandler

```
virtual
  ~IProgressIndicatorNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

IProgressIndicatorNotifyHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Processing

This function evaluates the event to determine if it is appropriate for this handler object to process.

dispatchHandlerEvent

If any of the following events are received, the progress indicator control observers are notified:

- Arm position changed
- Arm is dragged with the mouse
- Arm position moved to a tick
- Arm position moved to a pixel offset

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IPushButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
IPushButton

Inherited By IGraphicPushButton

Header File ipushbut.hpp

Members

Member	Page	Member	Page
Constructor	776	hasBorder	775
addBorder	775	help	782
calcMinimumSize	781	isDefault	777
classDefaultStyle	782	isHelp	778
convertToGUIStyle	778	isSystemCommand	779
defaultButton	782	noBorder	782
defaultStyle	778	passEventToOwner	780
disableDefault	777	registerCallbacks	781
disableHelp	777	removeBorder	775
disableSystemCommand	779	setDefaultStyle	778
enableDefault	777	systemCommand	783
enableHelp	777	unregisterCallbacks	781
enableSystemCommand	779	~IPushButton	777

The IPushButton class creates and manages push button control windows.

The standard push button generates an ICommandEvent (p. 210). However, the application can change the window style value to generate a help request or system command event.

To change the push button event (message) processing, call either of the following:

- IPushButton::enableHelp (p. 777)
- IPushButton::enableSystemCommand (p. 779)

The preceding functions add or remove the following styles:

- help (p. 782)

© Copyright IBM Corp. 1992, 1996

- systemCommand (p. 783)

When the user selects a push button, the push button generates an ICommandEvent that is routed to its owner window, or it generates a help request. You can process an ICommandEvent with an ICommandHandler.

You can derive classes from the following handlers and attach them to a push button control:

- IKeyboardHandler (p. 490)
- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)

Motif The AIX release of the User Interface Class Library does not support dialog templates.

Public Functions

Borders

Use these members to query and modify the noBorder style of a push button object. By default, the push button is displayed with a border drawn around it.

addBorder Adds or removes the push button border.

```
virtual IPushButton&
  addBorder( Boolean add = true );
```

Win	PM	Motif
I	*Y*	*Y*

hasBorder If the push button has a border, true is returned. Otherwise, false is returned.

```
Boolean
  hasBorder() const;
```

Win	PM	Motif
I	*Y*	*Y*

removeBorder

Removes the border from the push button. This also prevents the background of the push button from being drawn with the color that is currently set for background drawing.

```
virtual IPushButton&
  removeBorder();
```

Win	PM	Motif
I	*Y*	*Y*

© IBM Corp. 1992, 1996

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IPushButton objects because both the copy constructor and assignment operator are private functions.

IPushButton

1
```
IPushButton( unsigned long id,
             IWindow* parent,
             IWindow* owner,
             const IRectangle& initial = IRectangle ( ),
             const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Creates a push button control and an object for it.

id A push button control ID.

parent The parent window.

owner The owner window.

initial The initial position and size of the control you are constructing. Optional.

style The control's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2
```
IPushButton( unsigned long id,
             IWindow* parent );
```

Win	PM	Motif
Y	Y	Y

Creates an object for a push button control that exists in a dialog window.

id The identifier of the existing push button control.
parent The parent window.

3
```
IPushButton( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

Creates an object for an existing push button control.

handle The window handle of an existing push button control.

~IPushButton

```
virtual
  ~IPushButton();
```

Win	PM	Motif
Y	Y	Y

Default Buttons

Use these members to query and modify the defaultButton style of a push button object. A default push button is the push button pressed when the user presses the Enter key.

disableDefault

Removes the style defaultButton (p. 782) from the push button.

```
virtual IPushButton&
  disableDefault();
```

Win	PM	Motif
Y	Y	Y

enableDefault

Adds or removes the style defaultButton (p. 782).

```
virtual IPushButton&
  enableDefault( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

isDefault

If the style defaultButton (p. 782) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isDefault() const;
```

Win	PM	Motif
Y	Y	Y

Help Buttons

Use these members to query and modify the help style of a push button object. A button with the help style enabled generates help events instead of command events when selected.

disableHelp

Removes the style help (p. 782) from the push button.

```
virtual IPushButton&
  disableHelp();
```

Win	PM	Motif
Y	Y	Y

enableHelp

Adds or removes the style help (p. 782).

```
virtual IPushButton&
  enableHelp( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

isHelp

If the style help (p. 782) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isHelp() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the push button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 782) unless you have changed the style using setDefaultStyle (p. 778).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent push buttons.

style Use the styles provided by IPushButton Styles (p. 782) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

System Command Buttons

Use these members to query and modify the systemCommand style of a push button object. A button with the systemCommand style enabled generates events that can be used by overriding ICommandHandler::systemCommand instead of ICommandHandler::command.

disableSystemCommand

Removes the style systemCommand (p. 783) from the push button.

```
virtual IPushButton&
  disableSystemCommand();
```

Win	PM	Motif
Y	Y	Y

enableSystemCommand

Adds or removes the style systemCommand (p. 783).

```
virtual IPushButton&
  enableSystemCommand( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

isSystemCommand

If the style systemCommand (p. 783) is set, true is returned. Otherwise, false is returned.

```
Boolean
  isSystemCommand() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IButton		
allowsMouseClickFocus	disableMouseClickFocus	highlight
backgroundColor	enableMouseClickFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup

© IBM Corp. 1992, 1996

IPushButton

IControl		
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IPushButton objects because both the copy constructor and assignment operator are private functions.

IPushButton

```
IPushButton();
```

Win	PM	Motif
N	*N*	*Y*

Creates an IPushButton object. This protected constructor is used by derived classes to create objects of this class. Derived classes that use this constructor must create the underlying GUI object.

Motif Creates an IPushButton object without an associated XmPushButton widget. This allows derived classes to create their own widgets.

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

passEventToOwner

Determines if the event is passed on to the owner.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	N	Y

registerCallbacks

Registers all possible callbacks to this object for events it might receive. IHandler derived classes later determine which events they will process.

If classes you derive override this function, the override must call Inherited::registerCallbacks to register the applicable callbacks.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

unregisterCallbacks

Removes any callbacks and X event handlers added by the registerCallbacks function.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Layout Support

Layout support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size this push button control can be, based on the text string length and the current font.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

© IBM Corp. 1992, 1996

Public Data

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the push button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

classDefaultStyle

Provides the original default style for this class, which is the following: IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

defaultButton

Specifies that this is a default push button. If the parent or owner window is a canvas or frame window, the **Enter** key can be used to select the push button without having the cursor on the push button. Visually, a default push button has a darker and thicker border.

```
static const Style
  defaultButton;
```

Win	PM	Motif
Y	*Y*	*Y*

help

Generates a help request, instead of a command event, when the push button is selected.

```
static const Style
  help;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The general help window for the owner window is displayed when the user selects a **Help** push button. This is in accordance with the *Motif Style Guide* for dialogs.

noBorder

Displays the push button without a border.

```
static const Style
  noBorder;
```

Win	PM	Motif
I	*Y*	*Y*

systemCommand

Generates a system command event, instead of a command event, when the user selects a push button. The only portable system command is ISystemMenu::idClose. This is the system command for closing a window. You can override the ICommandHandler::systemCommand function to process system command events.

```
static const Style
  systemCommand;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IButton		
buttonClickId	**noPointerFocus**	

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

© IBM Corp. 1992, 1996

IPushButton

IBase		
recoverable	**unrecoverable**	

Nested Classes

IPushButton contains the following nested classes:

IPushButton::Style (see page 785)

IPushButton::Style

Derivation

IBase
  IBitFlag
    IPushButton::Style

Inherited By None.

Header File ipushbut.hpp

The nested class IPushButton::Style provides a set of valid styles for the IPushButton (p. 774) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IRadioButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
ISettingButton
IRadioButton

Inherited By None.

Header File iradiobt.hpp

Members

Member	Page	Member	Page
Constructor	787	isAutoSelect	787
autoSelect	793	isCursorSelect	789
calcMinimumSize	792	noCursorSelect	793
classDefaultStyle	793	noSelections	793
convertToGUIStyle	790	registerCallbacks	791
defaultStyle	790	selectedIndex	789
disableAutoSelect	787	setDefaultStyle	790
disableCursorSelect	789	unregisterCallbacks	792
enableAutoSelect	787	~IRadioButton	788
enableCursorSelect	789		

The IRadioButton class creates and manages radio button control windows. A *radio button* is a circle or a diamond with associated text representing a choice. When a user selects the choice, the radio button is filled to indicate that the choice is selected. The user can clear the radio button by selecting another radio button. Radio buttons are mutually exclusive.

You can process a radio button's selection by deriving a handler from ISelectHandler (p. 851) and adding your handler either to the radio button or to its owner window.

Motif The IRadioButton constructor creates objects of this class using the XmToggleButton widget. IWindow::handle (p. 1049) returns the handle of the XmToggleButton widget.

The User Interface Class Library provides the behavior of an IRadioButton object via private callbacks. The library treats all radio buttons with the same parent as belonging to the same group.

© Copyright IBM Corp. 1992, 1996

Public Functions

Auto Select

Auto select members query and modify the autoSelect style of a radio button object. The autoSelect style determines whether the state of the radio button is automatically changed when the user clicks on it.

disableAutoSelect

Removes the style autoSelect (p. 793) from the radio button control. If auto select is disabled, the application is responsible for changing the state of the radio buttons when the radio button is clicked.

```
virtual IRadioButton&
  disableAutoSelect();
```

Win	PM	Motif
Y	*Y*	*I*

enableAutoSelect

Adds or removes the style autoSelect (p. 793).

```
virtual IRadioButton&
  enableAutoSelect( Boolean enable = true );
```

Win	PM	Motif
Y	*Y*	*I*

Motif The autoSelect style is always in effect.

isAutoSelect

If the radio button control has the style autoSelect (p. 793) set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isAutoSelect() const;
```

Win	PM	Motif
Y	*Y*	*I*

Motif Motif always uses autoSelect, and this function returns true.

Constructors

You can construct and destruct objects of the IRadioButton class. You cannot copy or assign IRadioButton objects because both the copy constructor and assignment operator are private functions.

IRadioButton

© IBM Corp. 1992, 1996

IRadioButton

1 `IRadioButton( unsigned long id,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial = IRectangle ( ),`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	Y	Y

Creates a radio button control and an object for it.

id A radio button control ID.

parent The parent window.

owner The owner window.

initial The initial position and size of the control you are constructing. Optional.

style The control's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2 `IRadioButton( unsigned long id,`
`IWindow* parent );`

Win	PM	Motif
Y	Y	Y

Creates an object for a radio button control that exists in a dialog window.

id The identifier of the existing radio button control.
parent The parent window.

3 `IRadioButton( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	Y

Creates an object for an existing radio button control.

handle The window handle of an existing radio button control.

~IRadioButton

```
virtual
  ~IRadioButton();
```

Win	PM	Motif
Y	Y	Y

Cursor Select

Cursor select members query and modify the noCursorSelect style of a radio button object. If enabled, the noCursorSelect style prevents the radio button from selecting itself when given the focus as a result of an arrow key or tab key.

disableCursorSelect

Adds the radio button style noCursorSelect (p. 793).

```
virtual IRadioButton&
  disableCursorSelect();
```

Win	PM	Motif
I	*Y*	*I*

enableCursorSelect

Removes or adds the radio button style noCursorSelect (p. 793). If removed, the radio button is cursor-selectable.

```
virtual IRadioButton&
  enableCursorSelect( Boolean enable = true );
```

Win	PM	Motif
I	*Y*	*I*

isCursorSelect

If the radio button is cursor-selectable, true is returned. Otherwise, false is returned.

```
Boolean
  isCursorSelect() const;
```

Win	PM	Motif
I	*Y*	*I*

Motif This function returns false.

Selection

Use these members to query the index of the selected radio button. The selection state of a button is typically changed by the user clicking on it using the mouse or pressing a key to select it. You can also change or query the state using User Interface Class Library functions.

selectedIndex

Returns the 0-based index of the selected radio button in a group. If no radio button is selected, noSelections is returned.

Note: The index returned includes all windows that are part of the group, not just radio buttons. To ensure that the index only includes radio buttons, be sure that the group does not contain any other windows.

```
unsigned long
  selectedIndex() const;
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

PM The index of any control in the group can be returned.

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the radio button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 793) unless you have changed the style using setDefaultStyle (p. 790).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent radio buttons.

style Use the styles provided by IRadioButton Styles (p. 792) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

ISettingButton		
deselect	enableAutoSelect	isAutoSelect
disableAutoSelect	enableNotification	isSelected

IButton		
allowsMouseClickFocus	disableMouseClickFocus	highlight
backgroundColor	enableMouseClickFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

registerCallbacks

Registers Motif callbacks for the widget created during object construction of this class.

© IBM Corp. 1992, 1996

IRadioButton

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	N	Y

unregisterCallbacks

Unregisters Motif callbacks for the widget created during object construction of this class.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size that this radio button control based on the text string length and the current font.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

ISettingButton		
passEventToOwner		

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Selection

Use these members to query the index of the selected radio button. The selection state of a button is typically changed by the user clicking on it using the mouse or pressing a key to select it. You can also change or query the state using User Interface Class Library functions.

noSelections The value returned by selectedIndex when there are no buttons selected in the group.

```
static const unsigned long
  noSelections;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the radio button styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IButton Styles (p. 139)

autoSelect Specifies a selection technique in which the user changes the current selection automatically by moving the keyboard cursor. If auto select is enabled, the state of the button automatically changes to the next appropriate state. For radio buttons, when auto select is enabled, all other buttons in the same group are deselected automatically when a button is selected. If auto select is disabled, the application must change the state of the button when it is clicked.

```
static const Style
  autoSelect;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The autoSelect style is always enabled in the Motif version.

classDefaultStyle

Provides the original default style for this class, which is the following: IRadioButton::autoSelect | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

noCursorSelect

Prevents the radio button from selecting itself when given the focus as a result of an arrow key or tab key.

```
static const Style
  noCursorSelect;
```

Win	PM	Motif
I	*Y*	*Y*

Motif The Motif version ignores the noCursorSelect style.

© IBM Corp. 1992, 1996

IRadioButton

Inherited Public Data

ISettingButton		
selectId		

IButton		
buttonClickId	noPointerFocus	

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IRadioButton contains the following nested classes:

IRadioButton::Style (see page 795)

IRadioButton::Style

Derivation

IBase
  IBitFlag
    IRadioButton::Style

Inherited By None.

Header File iradiobt.hpp

The nested class IRadioButton::Style provides a set of valid styles for the IRadioButton (p. 786) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IRecoordHandler

Derivation

IBase
IVBase
IHandler
IRecoordHandler

Inherited By None.

Header File irecohdr.hpp

Members

Member	Page	Member	Page
Constructor	797	windowSizeChanged	798
defaultHandler	797	~IRecoordHandler	797
dispatchHandlerEvent	798		

The IRecoordHandler class handles the notification that a window or control is being resized.

A recoordination event notifies a window when the size and position of the window's client area must be calculated. This event is processed by this handler to make any adjustments in child window coordinates which are necessary due to the window size change. This handler is distinct from an IResizeHander in that it runs before the IResizeHander and processes a different message. The message handled by the IRecoordHandler is WM_CALCVALIDRECTS/WM_NCCALCSIZE.

IRecoordHandler processes the resizing event by creating an IRecoordEvent object and routing it to the virtual windowSizeChanged function. You can override this virtual function to supply your own specialized processing of a resizing event.

The following return values from the virtual functions specify whether the window event is passed on for additional processing:

true The resizing event requires no additional processing. Do not pass it to another handler.

false The resizing event requires additional processing. Pass the resizing event to the next handler for additional processing, as follows:

- If there is another handler for the window, pass the resizing event to the next handler.
- If this is the last handler for the window, call IWindow::dispatch (p. 1082) to dispatch the event to the window's owner window.

© Copyright IBM Corp. 1992, 1996

- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the event.

Public Functions

Constructors

The only way to create objects of this class is by using the default constructor.

IRecoordHandler

Used by derived classes to construct objects of this class. This is the default constructor and accepts no parameters.

```
IRecoordHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

~IRecoordHandler

```
virtual
  ~IRecoordHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it. If it is, it calls the virtual function used to process the event.

defaultHandler

Returns a pointer to the default object of this class.

```
static IRecoordHandler*
  defaultHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

© IBM Corp. 1992, 1996

IRecoordHandler

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. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a resizing event is received, the windowSizeChanged virtual function is called.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing

Event-processing members must be supplied to process events affecting recoordination. You can override these virtual members in a derived class.

windowSizeChanged

Processes resizing events. This member is called when the event indicates that the size of the window with window handle *handle* has changed. It is implemented by derived classes.

```
virtual Boolean
  windowSizeChanged( IWindowHandle handle,
                     const IRectangle& newWindowRect,
                     const IRectangle& oldWindowRect );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IResizeEvent

Derivation

IBase
IVBase
IEvent
IResizeEvent

Inherited By None.

Header File isizeevt.hpp

Members

Member	Page	Member	Page
Constructor	800	oldSize	801
newSize	801	~IResizeEvent	800

The IResizeEvent class represents an event routed to a window notifying it of sizing changes (after the window's size changes). IResizeEvent provides functions to return the previous and new window sizes.

Motif The AIX release of the User Interface Class Library maps this event to the ConfigureNotify event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IResizeEvent Constructs an IResizeEvent object from the specified event. IResizeHandler::dispatchHandlerEvent (p. 804) constructs objects of this class from an object of the class IEvent (p. 304) and passes the resulting object to the function IResizeHandler::windowResize (p. 804).

```
IResizeEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IResizeEvent

```
virtual
  ~IResizeEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Size Information

A resize event describes the sizing change made to a window.

newSize Returns the size of the window after it is resized.

```
ISize
  newSize() const;
```

Win	PM	Motif
Y	Y	Y

oldSize Returns the size of the window before it is resized.

```
ISize
  oldSize() const;
```

Win	PM	Motif
Y	Y	Y

Win In Windows this function always returns ISize(0,0).

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**	

© IBM Corp. 1992, 1996

IResizeHandler

Derivation

IBase
IVBase
IHandler
IResizeHandler

Inherited By None.

Header File isizehdr.hpp

Members

Member	Page	Member	Page
Constructor	803	windowResize	804
dispatchHandlerEvent	804	~IResizeHandler	803

The IResizeHandler class handles the notification of resizing events. These events notify a window or control that it has increased or decreased on the screen.

Create a handler derived from IResizeHandler and attach it to a window or control. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the window or control to the resize handler.

When the resize handler receives a resize event, it creates an object of IResizeEvent (p. 800) and routes that object to the virtual function IResizeHandler::windowResize (p. 804). You can override this virtual function to supply your own specialized processing of a resize event.

The return value from the virtual function specifies whether the resize event should be passed on for additional processing, as follows:

true The resize event requires no additional processing. Do not pass it to another handler.

false Pass the resize event to the next handler for additional processing, as follows:

- If there is another handler for the control or window, pass the resize event to the next handler.
- If this is the last handler for the control or window, call IWindow::defaultProcedure (p. 1082) to process the resize event.

© Copyright IBM Corp. 1992, 1996

Public Functions

Constructors

Only derived classes can construct objects of this class.

~IResizeHandler

```
virtual
  ~IResizeHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

Only derived classes can construct objects of this class.

IResizeHandler

Derived classes call this default constructor to create objects of this class.

```
IResizeHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

© IBM Corp. 1992, 1996

IResizeHandler

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An IResizeHandler object processes only window-resizing events.

dispatchHandlerEvent

If a resize event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

A resize handler contains event-processing members that you use to process a window-resizing event. You should override at least one of these virtual functions in a derived class.

windowResize

Implemented by derived classes to handle a resizing event. A derived class must supply this function.

```
virtual Boolean
  windowResize( IResizeEvent& event ) = 0;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IResource

Derivation

IBase
IVBase
IResource

Inherited By

IPrivateResource
ISharedResource

Header File

ireslock.hpp

Members

Member	Page	Member	Page
Constructor	805	unlock	806
handle	807	~IResource	805
lock	806		

The IResource class is the abstract base resource class. Use a derived class to identify a serially reusable resource. These resources can be limited to the current process, as described by the IPrivateResource (p. 719) class, or they can be shared across multiple processes, as described by ISharedResource (p. 866) class.

Public Functions

Constructors

You cannot construct or destruct objects of this class. Derive a new class from this class to identify a serially reusable resource if the derived User Interface Class Library classes do not meet your needs.

IResource This function is the default constructor for this class.

```
IResource();
```

Win	PM	Motif
Y	Y	Y

~IResource

```
virtual
  ~IResource();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Resource Locking

Use resource-locking members to acquire or release a serialized access to this resource.

lock Acquires serialized access to the resource, preventing other threads of execution from accessing it. You can specify the value for *timeout* in milliseconds with -1 indicating an indefinite wait and 0 requesting an immediate return.

```
virtual IResource&
  lock( long timeOut = - 1 );
```

Win	PM	Motif
Y	Y	Y

timeOut Long value that represents in milliseconds how long to wait for this resource. The default value, -1, is an indefinite wait.

W32s Resource locks are not supported. No resources are acquired and locked when calling this member function.

Motif The X timer services used to implement the resource lock timeout have a resolution of one second. The lock function rounds up the specified timeout value to the next one-second interval.

unlock Releases access to the resource, allowing other threads of execution to access it.

```
virtual IResource&
  unlock();
```

Win	PM	Motif
Y	Y	Y

W32s Resource locks are not supported. No resources are unlocked and released when calling this member function.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Resource Handle

Use these members to obtain a handle to the resource.

handle Returns the handle for the resource.

```
virtual ISemaphoreHandle&
  handle() = 0;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IResourceId

Derivation IBase
IResourceId

Inherited By None.

Header File ireslib.hpp

Members

Member	Page	Member	Page
Constructor	808	id	810
asDebugInfo	809	operator unsigned long	810
asString	809	resourceLibrary	810

The IResourceId class is a wrapper class for a resource identifier. You can use the information contained in this class to determine the location of the resources if the resources reside in either an IResourceLibrary (p. 811) or an IDynamicLinkLibrary (p. 261). Most classes in the User Interface Class Library typically accept an IResourceId as input to describe the resources and those classes load the resources.

Public Functions

Constructors

You can construct objects of this class by providing a resource identifier, a resource identifier and a reference to an IResourceLibrary (p. 811), or a resource identifier and a reference to an IDynamicLinkLibrary (p. 261). You can also construct or initialize an object of this class from another IResourceId object.

IResourceId

1
```
IResourceId( unsigned long resourceId,
             const IResourceLibrary& resourceLibrary );
```

Win	PM	Motif
Y	Y	Y

resourceId Unsigned long value that represents the resource identifier.
resourceLibrary
Reference to the resource library where the resource resides.

Create an object to load the resource using the resource identifier and a reference to

© Copyright IBM Corp. 1992, 1996

an IResourceLibrary (p. 811). If you have already loaded the resource library that contains the resource, use this constructor.

2
```
IResourceId( unsigned long resourceId,
             const IDynamicLinkLibrary& dllLibrary );
```

Win	PM	Motif
Y	Y	N

resourceId Unsigned long value that represents the resource identifier.
dllLibrary Reference to the dynamic link library where the resource resides.

Create an object to load the resource using the resource identifier and a reference to an IDynamicLinkLibrary (p. 261). If have already loaded the dynamic link library that contains the resource, use this constructor.

3
```
IResourceId( unsigned long resourceId );
```

Win	PM	Motif
Y	Y	Y

resourceId Unsigned long value that represents the resource identifier.

Create an object to load the resource using only the resource identifier. The constructor uses the resource library returned by a call to ICurrentApplication::userResourceLibrary (p. 240).

Note: This is the recommended use of this class.

Diagnostics

Use these members for diagnostic purposes. They return an IString representation of an object of this class.

asDebugInfo Provides debugging information about the class object. It returns a string that contains the resource identifier as well as the results of a call to IResourceLibrary::asDebugInfo for its resource library.

```
IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString Provides textual information about the class object. It returns the resource identifier.

```
IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Resources

Use these members to get the resource identifier or a reference to the resource library object for this class.

© IBM Corp. 1992, 1996

IResourceId

id

Returns the identifier used for the resource.

```
unsigned long
  id() const;
```

Win	PM	Motif
Y	Y	Y

operator unsigned long

Returns the identifier used for the resource. Using this operator is the same as calling IResourceId::id (p. 810).

```
operator unsigned long() const;
```

Win	PM	Motif
Y	Y	Y

resourceLibrary

Returns a const reference to the resource library.

```
const IResourceLibrary&
  resourceLibrary() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IResourceLibrary

Derivation

IBase
  IVBase
    IResourceLibrary

Inherited By

IDynamicLinkLibrary

Header File

ireslib.hpp

Members

Member	Page	Member	Page
Constructor	812	loadMenu	816
asDebugInfo	812	loadMessage	817
asString	812	loadPointer	817
fileName	813	loadString	817
handle	813	operator =	812
isOpen	813	sizeBitmapTo	817
loadAccelTable	813	tryToLoadBitmap	817
loadBitmap	814	tryToLoadIcon	818
loadDialog	814	tryToLoadMessage	819
loadHelpTable	815	tryToLoadString	819
loadIcon	815	~IResourceLibrary	812

The IResourceLibrary class loads most of the system resources. The default location for the resources is the executable file. Use the IDynamicLinkLibrary (p. 261) class to load resources that are stored in a dynamic link library.

Typically, you only need to use this class when constructing objects of the IResourceId (p. 808) class. You can also use it to obtain either the User Interface Class Library's resource library or the default user resource library using ICurrentApplication::resourceLibrary (p. 238) or ICurrentApplication::userResourceLibrary (p. 240), respectively.

Public Functions

Constructors

Use these members to construct, copy, assign, and destruct objects of this class. You can construct objects of this class by using the default constructor, which does not accept any parameters, or by using the copy constructor to copy one resource library to another.

© Copyright IBM Corp. 1992, 1996

IResourceLibrary

IResourceLibrary

1 `IResourceLibrary();`

Win	PM	Motif
Y	Y	Y

Creates a resource library object using this, the default constructor.

2 `IResourceLibrary( const IResourceLibrary& resLibrary );`

Win	PM	Motif
Y	Y	Y

resourceLibrary
Reference to an existing resource library object.

Creates a resource library object using a reference to an existing resource library object. This is commonly known as a copy constructor.

operator = Assigns the member data of an object of this class to another object of this class.

```
IResourceLibrary&
  operator =( const IResourceLibrary& resLibrary );
```

Win	PM	Motif
Y	Y	Y

resLibrary Reference to an existing resource library object.

~IResourceLibrary

```
virtual
  ~IResourceLibrary();
```

Win	PM	Motif
Y	Y	Y

Diagnostics

Use these members for diagnostic purposes. They return an IString representation of an object of this class.

asDebugInfo Provides debugging information about the class object. It returns a string that contains the file name of the resource library.

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString Provides textual information about the class object. It returns a string that contains the file name of the resource library.

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Resource Information

Use these members to query general resource information.

fileName Returns the fully qualified file name of the resource library.

```
virtual IString
  fileName() const;
```

Win	PM	Motif
Y	Y	Y

handle Returns the handle that is used to load the resources.

```
virtual IModuleHandle
  handle() const;
```

Win	PM	Motif
Y	Y	Y

PM This is always 0 for an IResourceLibrary object.

isOpen Returns the open state of the resource file.

Note: *True* is always returned for an object of this class.

```
virtual Boolean
  isOpen() const;
```

Win	PM	Motif
Y	Y	Y

Resource Loading

Use these members to load specific resources from a resource file.

loadAccelTable

Loads an accelerator table from a resource file.

1
```
virtual void
  loadAccelTable( unsigned long accelTableId,
                  const IWindowHandle& menu ) const;
```

Win	PM	Motif
N	N	Y

accelTableId
Unsigned long value that represents the accelerator table identifier.

menu Reference to the handle of the window where the menu exists.

2
```
virtual IAccelTblHandle
  loadAccelTable( unsigned long accelTableId ) const;
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

accelTableId
Unsigned long value that represents the accelerator table identifier.

Exceptions	
IAccessError	The accelerator table was not loaded. The accelerator table identifier may be invalid.

loadBitmap Loads a bitmap from a resource file.

If the *cached* argument is true, the bitmap is reference-counted. Therefore, if multiple requests are made for the bitmap, the reference count is incremented and the same copy of the bitmap is used. When all of the references to the bitmap are deleted, the User Interface Class Library deletes the bitmap.

If you set the *cached* argument to false, you create a new copy of the bitmap that is never deleted.

1

```
virtual IBitmapHandle
  loadBitmap( unsigned long bitmapId,
              const ISize& bitmapSize,
              Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

bitmapId Unsigned long value that represents the bitmap identifier.

bitmapSize Reference to a size object that determines at which size the bitmap is loaded.

cached Boolean value that determines caching.

Use this version of the function if you want to specify the size at which the bitmap is loaded.

Exceptions	
IAccessError	The bitmap was not loaded. The bitmap identifier may be invalid.

2

```
virtual IBitmapHandle
  loadBitmap( unsigned long bitmapId,
              Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

bitmapId Unsigned long value that represents the bitmap identifier.
cached Boolean value that determines caching.

loadDialog Loads a dialog resource, creates a frame window, and sets the dialog as the client of the frame window.

```
virtual IWindowHandle
  loadDialog( unsigned long dialogId,
              IWindow* dialogParent,
              IWindow* dialogOwner,
              IWinProc* dialogProcedure,
              void* dialogCreateParameters ) const;
```

Win	PM	Motif
Y	Y	Y

dialogId Unsigned long value that represents the dialog identifier.

dialogParent
Pointer to a window object that is the dialog's parent.

dialogOwner
Pointer to a window object that is the dialog's owner.

dialogProcedure
Pointer to a dialog procedure for the dialog you are loading.

dialogCreateParameters
Pointer to an application-defined data area. The first 2 bytes of the data referenced by the pointer should be the total size of the data.

Motif AIX does not support dialogs.

Exceptions	
IAccessError	The dialog was not loaded. The dialog identifier, parent, owner, procedure, or create parameters may be invalid.

loadHelpTable

Loads a help table from a resource file. The IHelpWindow (p. 442) class provides more information on help windows.

```
virtual IResourceLibrary&
  loadHelpTable( IWindow* helpInstance,
                 unsigned long helpTableId ) const;
```

Win	PM	Motif
Y	Y	Y

helpInstance
Pointer to the window object for the help instance.

helpTableId
Unsigned long value that represents the help table identifier.

Exceptions	
IAccessError	The help table was not loaded. The help table identifier or the help table instance may be invalid.

loadIcon

Loads an icon from a resource file.

If the *cached* argument is true, the icon is reference-counted. Therefore, if multiple requests are made for the icon, the reference count is incremented and the same copy

© IBM Corp. 1992, 1996

of the icon is used. When all of the references to the icon are deleted, the User Interface Class Library deletes the icon.

If you set the *cached* argument to false, you create a new copy of the icon that is never deleted.

```
virtual IPointerHandle
  loadIcon( unsigned long iconId,
            Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

iconId Unsigned long value that represents the icon identifier.
cached Boolean value that determines caching.

Exceptions	
IAccessError	The icon was not loaded. The icon identifier may be invalid.

loadMenu

Loads a menu resource from a resource file.

1

```
virtual void
  loadMenu( unsigned long menuId,
            const IWindowHandle& menuHandle,
            IMenu* parentMenu ) const;
```

Win	PM	Motif
N	N	Y

menuId Unsigned long value that represents the menu identifier.
menuHandle
Reference to the handle of the menu owner window.
parentMenu
Pointer to a menu object that is the menu's parent.

Motif This member function calls a private IMenuBld class to read the menu resource from the Motif resource database, parses it, and calls the appropriate IMenu member function to create the menu items under *parentMenu*.

2

```
virtual IWindowHandle
  loadMenu( unsigned long menuId,
            IWindow* menuOwner ) const;
```

Win	PM	Motif
Y	Y	N

menuId Unsigned long value that represents the menu identifier.
menuOwner
Pointer to a window object that is the menu's owner.

Exceptions	
IAccessError	The menu was not loaded. The menu identifier or the menu owner may be invalid.

loadMessage

Loads a message resource from a resource file using the specified message identifier.

```
virtual IString
  loadMessage( unsigned long messageId ) const;
```

Win	PM	Motif
Y	Y	Y

messageId Unsigned long value that represents the message identifier.

loadPointer

Loads a pointer from a resource file.

If the *cached* argument is true, the pointer is reference-counted. Calling this function is the same as calling IResourceLibrary::loadIcon (p. 815).

```
virtual IPointerHandle
  loadPointer( unsigned long iconId,
               Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

iconId Unsigned long value that represents the icon identifier.
cached Boolean value that determines caching.

loadString

Loads a string resource from a resource file using the specified string identifier.

```
virtual IString
  loadString( unsigned long stringId ) const;
```

Win	PM	Motif
Y	Y	Y

stringId Unsigned long value that represents the string identifier.

sizeBitmapTo

Scales the bitmap to the size specified by the *newSize* parameter.

```
static IBitmapHandle
  sizeBitmapTo( const IBitmapHandle& currentHandle,
                const ISize& newSize );
```

Win	PM	Motif
N	N	Y

currentHandle
Reference to the handle of the bitmap to resize.
newSize Reference to a size object that contains the new bitmap size.

tryToLoadBitmap

Attempts to load a bitmap from a resource file. If this function cannot load the bitmap, it returns an IBitmapHandle with a 0 handle value rather than throwing an exception.

© IBM Corp. 1992, 1996

If the *cached* argument is true, the bitmap is reference-counted. Therefore, if multiple requests are made for the bitmap, the reference count is incremented and the same copy of the bitmap is used. When all of the references to the bitmap are deleted, the User Interface Class Library deletes the bitmap.

If you set the *cached* argument to false, you create a new copy of the bitmap that is never deleted.

1

```
virtual IBitmapHandle
  tryToLoadBitmap( unsigned long bitmapId,
                   const ISize& bitmapSize,
                   Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

bitmapId Unsigned long value that represents the bitmap identifier.

bitmapSize Reference to a size object that determines at which size the bitmap is loaded.

cached Boolean value that determines caching.

Use this version of the function if you want to specify the size at which the bitmap is loaded.

2

```
virtual IBitmapHandle
  tryToLoadBitmap( unsigned long bitmapId,
                   Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

bitmapId Unsigned long value that represents the bitmap identifier.
cached Boolean value that determines caching.

tryToLoadIcon

Attempts to load an icon from a resource file. If this function cannot load the icon, it returns an IPointerHandle with a 0 handle value rather than throwing an exception.

If the *cached* argument is true, the icon is reference-counted. Therefore, if multiple requests are made for the icon, the reference count is incremented and the same copy of the icon is used. When all of the references to the icon are deleted, the User Interface Class Library deletes the icon.

If you set the *cached* argument to false, you create a new copy of the icon that is never deleted.

```
virtual IPointerHandle
  tryToLoadIcon( unsigned long iconId,
                 Boolean cached = true ) const;
```

Win	PM	Motif
Y	Y	Y

iconId Unsigned long value that represents the icon identifier.
cached Boolean value that determines caching.

tryToLoadMessage

Attempts to load a message resource from a resource file. If this function cannot load the message, it returns an empty IString rather than throwing an exception.

```
virtual IString
  tryToLoadMessage( unsigned long messageId ) const;
```

Win	PM	Motif
Y	Y	Y

messageId Unsigned long value that represents the message identifier.

tryToLoadString

Attempts to load a string resource from a resource file. If this function cannot load the string, it returns an empty IString rather than throwing an exception.

```
virtual IString
  tryToLoadString( unsigned long stringId ) const;
```

Win	PM	Motif
Y	Y	Y

stringId Unsigned long value that represents the string identifier.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IResourceLock

Derivation

IBase
IVBase
IResourceLock

Inherited By None.

Header File ireslock.hpp

Members

Member	Page	Member	Page
Constructor	820	setLock	822
clearLock	821	~IResourceLock	821

The IResourceLock class locks a resource for a specified period of time.

A simple yet effective use of this class is to declare a local object of this class that is in scope for the period of time that the resource needs to be locked. The lock is automatically removed when the block of code is exited, thereby forcing the object out of scope.

If the specified period of time is reached before the resource can be acquired, an IResourceExhausted (Vol. I) exception is thrown.

W32s Win32s is a single threaded system. As a result, resource locks are not supported. Instances of the IResourceLock class can be created and the member functions can be called, but no system resource locks are obtained.

Public Functions

Constructors

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

IResourceLock

The only way to construct objects of this class is with this constructor. You specify a reference to an IResource (p. 805) and an optional value for the *timeOut* parameter. You can specify *timeOut* in milliseconds with -1 indicating an indefinite wait and 0 requesting an immediate return.

© Copyright IBM Corp. 1992, 1996

```
IResourceLock( IResource& resource,
               long timeOut = - 1 );
```

Win	PM	Motif
Y	Y	Y

resource Reference to the resource being locked.
timeOut Long value that represents in milliseconds how long to wait for the lock request. The default value, -1, is an indefinite wait.

W32s Resource locks are not supported. No resources are created when calling this constructor.

Motif The AIX timer services used to implement the resource lock *timeOut* have a resolution of one second. The timeout values you provide are rounded up to the next one-second interval.

~IResourceLock

```
virtual
  ~IResourceLock();
```

Win	PM	Motif
Y	Y	Y

W32s Resource locks are not supported. No resources are released when calling this destructor.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Resource Locking

Use these members to acquire or release a lock on the resource.

clearLock Releases the resource.

```
virtual IResourceLock&
  clearLock();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

W32s Resource locks are not supported. No resources are cleared when calling this member function.

Exceptions	
IAccessError	The resource object was not released. The semaphore handle may be invalid, there may be too many release requests, the release request was interrupted, or the process which owns the semaphore died.

setLock Acquires the resource. You can specify the value for *timeOut* in milliseconds with -1, indicating an indefinite wait, and 0, requesting an immediate return.

The constructor calls this function to lock the resource; IResourceLock::clearLock (p. 821) can then be called to clear the lock, and this function can be called to reset it. This is not the normal use of this class. Normally, you use the constructor and scope it to the instruction that requires serialization.

```
virtual IResourceLock&
  setLock( long timeOut = - 1 );
```

Win	PM	Motif
Y	Y	Y

timeOut Long value that represents in milliseconds how long to wait for the lock request. The default value, -1, is an indefinite wait.

W32s Resource locks are not supported. No resources are acquired when calling this member function.

Motif The AIX timer services used to implement the resource lock *timeOut* have a resolution of one second. This function rounds up the specified timeout value to the next one-second interval.

Exceptions	
IOutOfSystemResource	The resource object was not acquired. The available semaphores may be exhausted because the request for a lock timed out.
IAccessError	The resource object was not acquired. The semaphore handle may be invalid, there may be too many lock requests, the lock request was interrupted, or the process which owns the semaphore died.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IScrollBar

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IScrollBar

Inherited By None.

Header File iscroll.hpp

Members

Member	Page	Member	Page
Constructor	825	scrollableRange	828
autoSize	835	scrollBoxPosition	828
calcMinimumSize	835	scrollBoxPositionId	835
classDefaultStyle	836	scrollBoxRange	828
convertToGUIStyle	831	setDefaultStyle	832
defaultStyle	831	setForegroundColor	825
enableNotification	829	setHiliteForegroundColor	825
foregroundColor	824	setMinScrollIncrement	830
hiliteForegroundColor	824	setPageScrollIncrement	831
horizontal	836	setPrevScrollBoxPosition	828
isHorizontal	830	setScrollableRange	829
isVertical	830	setScrollBar	829
minScrollIncrement	830	setVisibleCount	829
moveScrollBoxTo	827	size	833
moveSizeTo	833	systemScrollBarWidth	832
pageScrollIncrement	830	systemScrollBoxLength	832
passEventToOwner	834	systemScrollButtonLength	832
position	833	unregisterCallbacks	834
prevScrollBoxPosition	827	vertical	836
registerCallbacks	834	visibleCount	829
resetForegroundColor	824	~IScrollBar	827
resetHiliteForegroundColor	824		

The IScrollBar class creates and manages scroll bar control windows.

PM The size limitation for a scroll bar is 32768.

Public Functions

© Copyright IBM Corp. 1992, 1996

Colors

Use these members to query, set, and reset the colors of the scroll bar.

foregroundColor

Returns the color of the shaft area.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Returns the default color of the shaft area.

This member is overridden in this derived class for specific operating system behavior.

hiliteForegroundColor

Returns the color of the scroll box area.

```
virtual IColor
  hiliteForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*Y*

Returns the default color of the scroll box area.

This member is overridden in this derived class for specific operating system behavior.

resetForegroundColor

Resets the color of the shaft area.

```
virtual IScrollBar&
  resetForegroundColor();
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

resetHiliteForegroundColor

Resets the color of the scroll box color area.

```
virtual IScrollBar&
  resetHiliteForegroundColor();
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

setForegroundColor

Sets the color of the shaft area.

color Color object used to set the shaft area.

```
virtual IScrollBar&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
N	N	Y

setHiliteForegroundColor

Sets the color of the scroll box area.

color Color object used to set the scroll box area.

```
virtual IScrollBar&
  setHiliteForegroundColor( const IColor& color );
```

Win	PM	Motif
N	N	Y

This member is overridden in this derived class for specific operating system behavior.

Constructors

You can construct and destruct objects of this class.

IScrollBar

1
```
IScrollBar( const IWindowHandle& handle );
```

Win	PM	Motif
Y	Y	Y

Use this function to construct objects of this class using the handle of an existing scroll bar control.

handle The window handle of an existing scroll bar control.

2
```
IScrollBar( unsigned long id,
            IWindow* parent,
            IWindow* owner,
            const IRectangle& initial = IRectangle ( ),
            const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Use this function to construct objects of this class using the parent window, owner window, optional size and location, and optional style parameters.

The characteristics of the scroll box from this constructor default to the following:

© IBM Corp. 1992, 1996

IScrollBar

Range of items 100
Visible items 25
Position 1

windowId The scroll bar control ID.

parent The parent window.

owner The owner window.

initial A rectangle for the scroll box control. It specifies the initial position and size of the scroll box you construct. Optional.

style The initial style for the control. The default is classDefaultStyle (p. 836). Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

3

```
IScrollBar( unsigned long id,
            IWindow* parent,
            IWindow* owner,
            const IRange& scrollableItems,
            unsigned long visibleItemCount,
            unsigned long scrollBoxPosition = 1,
            const IRectangle& initial = IRectangle ( ),
            const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Use this function to construct objects of this class using the parent window, owner window, number of scrollable items, number of items displayed, initial position of the scroll box, optional size and location, and optional style parameters.

windowId The scroll bar control ID.

parent The parent window.

owner The owner window.

scrollableItems
The range of all scrollable items.

visibleItemCount
The number of items to display.

scrollBoxPosition
The initial position of the scroll box within the scroll bar. The default is 1. Optional.

initial A rectangle for the scroll box control. It specifies the initial position and size of the scroll box you construct. Optional.

style The initial style for the control. The default is classDefaultStyle (p. 836). Optional.

PM The minimum range you can set is 0 and the maximum is 32767.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

4
```
IScrollBar( unsigned long id,
            IWindow* parent );
```
Win	PM	Motif
Y	Y	Y

Use this function to construct objects of this class using the parent window.

windowId The scroll bar control ID.
parent The parent window.

~IScrollBar

```
virtual
  ~IScrollBar();
```
Win	PM	Motif
Y	Y	Y

Manipulation Operations

Use these members to change and query how the scroll bar operates.

moveScrollBoxTo

Moves the scroll box to the specified position.

```
virtual IScrollBar&
  moveScrollBoxTo( unsigned long firstItem = 1 );
```
Win	PM	Motif
Y	Y	Y

prevScrollBoxPosition

Returns the previous position of the scroll box.

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

© IBM Corp. 1992, 1996

IScrollBar

scrollableRange

Returns the range that the scroll bar represents.

```
virtual IRange
  scrollableRange() const;
```

Win	PM	Motif
Y	Y	Y

scrollBoxPosition

Returns the current position of the scroll box.

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

Win	PM	Motif
Y	Y	Y

scrollBoxRange

Returns the range in which the scroll box can be positioned.

See class IRange (Vol. I) for more information.

```
virtual IRange
  scrollBoxRange() const;
```

Win	PM	Motif
Y	Y	Y

Win In this environment, the scroll box (thumb) size is variable. The range returned by this function is adjusted to account for the scroll box size. This is done by subtracting the visible count (scroll box size) from the scrollable range.

W32s In this environment, the scroll box size is fixed. Therefore, the range returned by this function is not adjusted to account for a variable scroll box (thumb) size.

setPrevScrollBoxPosition

Sets the value of the previous scroll box position.

value Number representing the control's previous position.

```
virtual IScrollBar&
  setPrevScrollBoxPosition( unsigned long value );
```

Win	PM	Motif
N	N	Y

Motif The User Interface Class Library application developer should never have to call this function. IScrollEvent needs the previous position of the scroll bar to calculate how far the scroll bar has moved. When an IScrollEvent is generated in Motif, the scroll bar has already been moved to the new position by the XmScrollBar widget. On the other hand, the OS/2 operating system generates the IScrollEvent before the scroll box has been moved. To calculate the amount the scroll bar has moved, the previous scroll box position must be saved.

setScrollableRange

Sets the range of all items the scroll bar can scroll to.

```
virtual IScrollBar&
  setScrollableRange( const IRange& minMax );
```

Win	PM	Motif
Y	Y	Y

PM The minimum range you can set is 0 and the maximum is 32767.

setScrollBar

Sets the range of all items and the number of items that are displayed. This is functionally equivalent to calling both setScrollableRange (p. 829) and setVisibleCount (p. 829).

```
virtual IScrollBar&
  setScrollBar( const IRange& scrollableRange,
                unsigned long visibleCount );
```

Win	PM	Motif
Y	Y	Y

PM The minimum range you can set is 0 and the maximum is 32767.

setVisibleCount

Defines the amount of the scrollable range that is displayed. Use this to determine the length of the scroll box and the default value of the page-scrolling increment.

```
virtual IScrollBar&
  setVisibleCount( unsigned long scrollableRangeUnits );
```

Win	PM	Motif
Y	Y	Y

visibleCount

Returns the amount of the scrollable range that is displayed. Use this to determine the length of the scroll box and the default value of the page-scrolling increment.

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

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables the scroll bar control to send notifications to any observer objects.

```
virtual IScrollBar&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Orientation

Use these members to query the orientation of the scroll bar.

isHorizontal If the scroll bar is horizontal, true is returned. Otherwise, false is returned.

```
Boolean
  isHorizontal() const;
```

Win	PM	Motif
Y	Y	Y

isVertical If the scroll bar is vertical, true is returned. Otherwise, false is returned.

```
Boolean
  isVertical() const;
```

Win	PM	Motif
Y	Y	Y

Scrolling Increment

Use these members to query and set the scrolling increments for the scroll bar. These values determine the scroll amounts stored in IScrollEvent objects.

minScrollIncrement

Returns the amount of the scrollable range that is scrolled by selecting the scroll buttons.

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

Win	PM	Motif
Y	Y	Y

pageScrollIncrement

Returns the amount of the scrollable range that is scrolled by selecting the scroll shaft.

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

Win	PM	Motif
Y	Y	Y

setMinScrollIncrement

Sets the amount of the scrollable range that is scrolled by selecting the scroll buttons.

```
virtual IScrollBar&
  setMinScrollIncrement(
   unsigned long scrollableRangeUnits = 1);
```

Win	PM	Motif
Y	Y	Y

setPageScrollIncrement

Sets the amount that a page scrolls when a user selects the scroll shaft.

scrollableRangeUnits

The default is 0. Using this default sets the default page-scrolling increment to visible count minus the minimum scroll increment.

For example, if a window contains 20 lines of text, the visible count is 20. If the minimum scroll increment is one line, the default page scroll increment is 19 (20-1).

If the visible count and the minimum scroll increment are the same, this function uses the minimum scroll increment. Otherwise, the page does not scroll.

For example, if the visible count is 20 and the minimum scrolling increment is 20, the preceding formula sets the default page-scrolling increment to 0, which prevents the page from scrolling. The User Interface Class Library avoids this possibility by setting the default page-scrolling increment to the minimum scroll increment in these situations.

```
virtual IScrollBar&
  setPageScrollIncrement(
   unsigned long scrollableRangeUnits = 0);
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the scroll bar styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	*Y*	*N*

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 836) unless you have changed the style using setDefaultStyle (p. 832).

© IBM Corp. 1992, 1996

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent scroll bars.

style Use the styles provided by IScrollBar Styles (p. 835) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

System Values

These static members return various system sizes. For these members, "length" refers to the size in the direction of a scroll bar's orientation (that is, in the X-direction for a horizontal scroll bar and in the Y-direction for a vertical scroll bar). "Width" refers to the size in the direction perpendicular to the scroll bar's orientation (that is, in the Y-direction for a horizontal scroll bar and in the X-direction for a vertical scroll bar).

systemScrollBarWidth

Returns the system width of a vertical or horizontal scroll bar.

```
static unsigned long
  systemScrollBarWidth( Boolean verticalScrollBar = true );
```

Win	PM	Motif
Y	Y	Y

systemScrollBoxLength

Returns the system length of a scroll box for a vertical or horizontal scroll bar.

```
static unsigned long
  systemScrollBoxLength( Boolean verticalScrollBar = true );
```

Win	PM	Motif
Y	Y	Y

systemScrollButtonLength

Returns the system length of a scroll button for a vertical or horizontal scroll bar.

```
static unsigned long
  systemScrollButtonLength( Boolean verticalScrollBar = true );
```

Win	PM	Motif
Y	Y	Y

Window Positioning

Use these members to set and query the size and position of scroll bars. Unless otherwise noted, the orientation of the coordinates accepted and returned by these members is the application orientation. For more information about coordinate orientation, see ICoordinateSystem (p. 230).

moveSizeTo Changes the position and size of the scroll bar.

```
virtual IScrollBar&
  moveSizeTo( const IRectangle& rect );
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

position Returns the position of the scroll bar.

```
virtual IPoint
  position() const;
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

size Returns the size of the scroll bar.

```
virtual ISize
  size() const;
```

Win	PM	Motif
N	*N*	*Y*

This member is overridden in this derived class for specific operating system behavior.

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

passEventToOwner

Determines whether this event can be passed on to the owner of this control.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	*N*	*Y*

registerCallbacks

Registers all possible callbacks and X event handlers to this object for events it might receive. IHandler derived classes later determine which events they will process.

If classes you derive override the registerCallbacks member function, the override must call Inherited::registerCallbacks to register the applicable callbacks and X event handlers.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

unregisterCallbacks

Removes X-Motif callbacks that were registered for the XmScrollBar widget. IWindow (p. 1044) calls this function, so the User Interface Class Library application developer should not call this function directly.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the recommended minimum size of this scroll bar control.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

scrollBoxPositionId

Notification identifier provided to observers when the scroll box position of the scroll bar changes. IScrollBar provides the new scroll box position in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  scrollBoxPositionId;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the scroll bar styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

autoSize Causes automatic sizing of the scroll bar to occur. For a vertical scroll bar, the width is determined. For a horizontal scroll bar, the height is determined.

© IBM Corp. 1992, 1996

```
static const Style
  autoSize;
```

Win	PM	Motif
N	Y	Y

classDefaultStyle

Provides the original default style for this class, which is the following: IScrollBar::vertical | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

horizontal

Causes a horizontal scroll bar to be created. This style is ignored if it is used with vertical.

```
static const Style
  horizontal;
```

Win	PM	Motif
Y	Y	Y

vertical

Causes a vertical scroll bar to be created. This style is used if both it and horizontal are specified.

```
static const Style
  vertical;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IScrollBar contains the following nested classes:

IScrollBar::Style (see page 838)

© IBM Corp. 1992, 1996

IScrollBar::Style

Derivation

IBase
IBitFlag
IScrollBar::Style

Inherited By None.

Header File iscroll.hpp

The nested class IScrollBar::Style provides a set of valid styles for the IScrollBar (p. 823) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IScrollBarNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        IScrollBarNotifyHandler

Inherited By None.

Header File isclnhdr.hpp

Members

Member	Page
Constructor	839
dispatchHandlerEvent	840
~IScrollBarNotifyHandler	839

The IScrollBarNotifyHandler class processes events for all classes of scroll bars.

This class is designed to handle events that require the scroll bar class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IScrollBarNotifyHandler

This is the default constructor and accepts no parameters.

```
IScrollBarNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~IScrollBarNotifyHandler

```
virtual
  ~IScrollBarNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

IScrollBarNotifyHandler

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 the scroll bar observers if the following event is received:

scrollBoxPosition event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IScrollEvent

Derivation

IBase
IVBase
IEvent
IScrollEvent

Inherited By None.

Header File iscrlevt.hpp

Members

Member	Page	Member	Page
Constructor	842	scrollBarWindow	843
newScrollBoxPosition	843	~IScrollEvent	842
scrollAmount	843		

The IScrollEvent class represents a scroll event for an IScrollBar (p. 823) object. This event is a notification for interactions with an IScrollBar object. An IScrollHandler (p. 845) object creates an IScrollEvent object and passes the event to one of its virtual functions for processing.

Public Functions

Constructors

You can construct and destruct objects of this class.

IScrollEvent Although you can construct objects of this class, typically IScrollHandler::dispatchHandlerEvent (p. 847) creates objects of this class from an object of the class IEvent (p. 304).

```
IScrollEvent( const IEvent& event );
```

Win	PM	Motif
Y	Y	Y

~IScrollEvent

```
virtual
  ~IScrollEvent();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Event Information

Use these members to query information related to the scroll event.

newScrollBoxPosition

Returns the new position of the scroll box. This is valid only for a scroll event passed to IScrollHandler::scrollBoxTrack (p. 849) or IScrollHandler::scrollBoxTrackEnd (p. 849).

```
unsigned long
  newScrollBoxPosition() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	This event is invalid for this function.

scrollAmount

Returns the number of units that the scroll event wants the scroll box to scroll. If a backward scroll event occurs, a negative value is returned.

```
long
  scrollAmount() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The scroll type is unknown; therefore, the distance scrolled cannot be determined.

scrollBarWindow

Returns a pointer to the IScrollBar (p. 823) object that the scroll event applies to.

```
IScrollBar*
  scrollBarWindow() const;
```

Win	PM	Motif
Y	Y	Y

Motif This function throws an IInvalidRequest exception if the window is not a derived class of XmScrollBar (an IScrollBar).

Inherited Public Functions

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle

© IBM Corp. 1992, 1996

IScrollEvent

IEvent		
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**	

IScrollHandler

Derivation

IBase
  IVBase
    IHandler
      IScrollHandler

Inherited By None.

Header File iscrlhdr.hpp

Members

Member	Page	Member	Page
Constructor	846	pageLeft	848
dispatchHandlerEvent	847	pageRight	848
lineDown	847	pageUp	848
lineLeft	847	scrollBoxTrack	849
lineRight	848	scrollBoxTrackEnd	849
lineUp	848	scrollEnd	849
moveScrollBox	849	~IScrollHandler	846
pageDown	848		

The IScrollHandler class handles events resulting from a user interacting with an IScrollBar (p. 823) object, such as when the user clicks on the down scroll button of a scroll bar.

Create a handler derived from IScrollHandler and attach it to a scroll bar's owner window. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the owner window to the scroll handler.

When the scroll handler receives a scroll event, it creates an IScrollEvent (p. 842) object and routes that object to the appropriate IScrollHandler virtual function. Override these virtual functions to supply your own specialized processing of a scroll event.

The return value from the virtual functions specifies whether the scroll event is passed on for additional processing, as follows:

true The scroll event requires no additional processing. Do not pass it to another handler.

© Copyright IBM Corp. 1992, 1996

false The scroll event requires additional processing. Pass the scroll event to the next handler for additional processing, as follows:

- If there is another handler for the scroll bar's owner window, pass the scroll event to the next handler.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the scroll event.

Scroll events are not dispatched to the IScrollBar object itself, but to the IScrollBar's owner window.

The Motif scroll bar has slightly different characteristics than the OS/2 and Windows scroll bars. The Presentation Manager and Windows scroll box does not automatically scroll to a new position when the user requests a change in position, unless the user drags the scroll box. The IScrollHandler can move the scroll box by calling moveScrollBox with the IScrollEvent object.

In contrast, the Motif version of the scroll bar uses the XmScrollBar widget, which automatically moves the scroll box to its new position before the event is dispatched to this handler.

You can obtain portable code by always calling moveScrollBox and using the IScrollEvent to determine the new position. The Presentation Manager and Windows versions move the scroll box to the new location and the Motif version moves the scroll box to its current position.

Public Functions

Constructors

You can construct and destruct objects of this class.

IScrollHandler

This is the default constructor and accepts no parameters.

```
IScrollHandler();
```

Win	PM	Motif
Y	Y	Y

~IScrollHandler

```
virtual
  ~IScrollHandler();
```

Win	PM	Motif
Y	Y	Y

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. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a scroll event is received, the appropriate virtual function is called.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members must be supplied to process a scroll event. You can override these virtual members in a derived class.

lineDown Scrolls down one line. Derived classes implement this function to process a scroll event that a user initiates by selecting the down scroll button of a vertical scroll bar.

```
virtual Boolean
  lineDown( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

lineLeft Scrolls left one line. Derived classes implement this function to process a scroll event that a user initiates by selecting the left scroll button of a horizontal scroll bar.

© IBM Corp. 1992, 1996

```
virtual Boolean
  lineLeft( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

lineRight Scrolls right one line. Derived classes implement this function to process a scroll event that a user initiates by selecting the right scroll button of a horizontal scroll bar.

```
virtual Boolean
  lineRight( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

lineUp Scrolls up one line. Derived classes implement this function to process a scroll event that a user initiates by selecting the up scroll button of a vertical scroll bar.

```
virtual Boolean
  lineUp( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

pageDown Scrolls down one page. Derived classes implement this function to process a scroll event that a user initiates by selecting the scroll shaft below the scroll box of a vertical scroll bar.

```
virtual Boolean
  pageDown( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

pageLeft Scrolls left one page. Derived classes implement this function to process a scroll event that a user initiates by selecting the scroll shaft to the left of the scroll box of a horizontal scroll bar.

```
virtual Boolean
  pageLeft( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

pageRight Scrolls right one page. Derived classes implement this function to process a scroll event that a user initiates by selecting the scroll shaft to the right of the scroll box of a horizontal scroll bar.

```
virtual Boolean
  pageRight( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

pageUp Scrolls up one page. Derived classes implement this function to process a scroll event that a user initiates by selecting the scroll shaft above the scroll box of a vertical scroll bar.

```
virtual Boolean
  pageUp( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

scrollBoxTrack

Tracks the movement of the scroll box. Derived classes implement this function to process a scroll event that a user initiates by dragging the scroll box with the mouse.

```
virtual Boolean
  scrollBoxTrack( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

scrollBoxTrackEnd

Ends the tracking of scroll box movement. Derived classes implement this function to process the end of a scroll event. Tracking ends when the user stops dragging the scroll box with the mouse.

```
virtual Boolean
  scrollBoxTrackEnd( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

scrollEnd

Ends a scroll event. Derived classes implement this function to process the end of a scroll event not caused by moving the scroll box. For example, the user releases the mouse button after clicking on a scroll button or scroll shaft.

```
virtual Boolean
  scrollEnd( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

Implementation

These members are called by many of the virtual functions to provide some default processing of a scroll event.

moveScrollBox

Positions the scroll box based on the scroll event.

```
virtual IScrollHandler&
  moveScrollBox( IScrollEvent& event );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

© IBM Corp. 1992, 1996

IScrollHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ISelectHandler

Derivation

IBase
  IVBase
    IHandler
      ISelectHandler

Inherited By None.

Header File iselhdr.hpp

Members

Member	Page	Member	Page
Constructor	853	selected	854
dispatchHandlerEvent	853	~ISelectHandler	852
enter	853		

The ISelectHandler-derived classes handle events resulting from a user interacting with a selectable item, such as clicking a radio button. ISelectHandler objects process item selection events for the following controls:

ICheckBox (p. 146)
IComboBox (p. 182)
IContainerControl (Vol. III)
IListBox (p. 495)
IRadioButton (p. 786)

You can create a handler derived from ISelectHandler and attach it to a control that generates a selection event or to the control's owner window by calling IHandler::handleEventsFor (p. 413).

When the selection handler receives a selection event, it creates an IControlEvent (p. 227) object and routes that object to the appropriate ISelectHandler virtual function. You override these virtual functions to supply customized processing of a selection event.

The return value from the virtual functions specifies whether the selection event is be passed on for additional processing, as follows:

true The selection event requires no additional processing. The IWindow event dispatcher should not pass the event to another handler.

© Copyright IBM Corp. 1992, 1996

false Pass the selection event to the next handler for additional processing, as follows:

- If there is another handler for the control, pass the selection event to the next handler.
- For Presentation Manager applications, if this is the last handler for the control, call IWindow::dispatch (p. 1082) to dispatch the selection event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the selection event.

PM You can use ISelectHandler to process selection events for the class I3StateCheckBox (p. 8).

The ISelectHandler is called when you call IButton::click (p. 137). Calling ISettingButton::select (p. 859) does not cause an ISelectHandler to be called.

Motif The ISelectHandler is called when ISettingButton::select (p. 859) and IButton::click (p. 137) are called for radio button and check box controls.

Public Functions

Constructors

Only derived classes can construct objects of this class.

~ISelectHandler

```
virtual
  ~ISelectHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

Only derived classes can construct objects of this class.

ISelectHandler

Derived classes call this default constructor to create objects of this class.

```
ISelectHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An ISelectHandler object processes only selection events.

dispatchHandlerEvent

If a selection event is received, this function calls the appropriate virtual function.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Event Processing Functions

A *select handler* allows you to process a selection or enter action through its event-processing members. You should override at least one of these virtual functions in a derived class.

enter Implemented by derived classes to process an enter action. An enter action is caused by the user double clicking on a button, double clicking or pressing the Enter key on a list box, container, or combination box, or selecting an item from the drop-down list box of a combination box.

For buttons, this function calls the function selected (p. 854).

© IBM Corp. 1992, 1996

ISelectHandler

```
virtual Boolean
  enter( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

selected Implemented by derived classes to process a selection action.

```
virtual Boolean
  selected( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ISemaphoreHandle

Derivation

IBase
  IHandle
    ISemaphoreHandle

Inherited By None.

Header File ibhandle.hpp

Members

Member	Page
Constructor	855
operator Value	855

The ISemaphoreHandle class accesses semaphores. An example of a semaphore is a flag in a multiple-user application that prevents simultaneous access to a file.

Public Functions

Constructors

You can construct objects of this class.

ISemaphoreHandle

You can construct objects of this class from a semaphore handle (a value of type IHandle::Value), which defaults to 0.

```
ISemaphoreHandle( Value hsem = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

```
operator Value() const;
```

Win	PM	Motif
N	*N*	*Y*

© Copyright IBM Corp. 1992, 1996

ISemaphoreHandle

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

ISettingButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
ISettingButton

Inherited By

I3StateCheckBox
ICheckBox
IRadioButton

Header File isetbut.hpp

Members

Member	Page	Member	Page
Constructor	858	isSelected	859
deselect	859	passEventToOwner	860
disableAutoSelect	858	select	859
enableAutoSelect	858	selectId	861
enableNotification	859	~ISettingButton	858
isAutoSelect	858		

The ISettingButton class is an abstract base class for the following control window classes:

- I3StateCheckBox (p. 8)
- ICheckBox (p. 146)
- IRadioButton (p. 786)

Public Functions

Auto Select

The auto select setting determines whether the state of the button is automatically changed when the user clicks on it. If auto select is enabled, the state of the button automatically changes to the next appropriate state. If auto select is disabled, the application must change the state of the button when it is clicked.

© Copyright IBM Corp. 1992, 1996

disableAutoSelect

Disables the automatic selection style for a button.

```
virtual ISettingButton&
  disableAutoSelect() = 0;
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function; therefore, it has no effect.

enableAutoSelect

Enables the automatic selection style for a button.

```
virtual ISettingButton&
  enableAutoSelect( Boolean enable ) = 0;
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function; therefore, it has no effect.

isAutoSelect

If the automatic selection style for a button is set on, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isAutoSelect() const = 0;
```

Win	PM	Motif
Y	Y	Y

Motif The AIX version is always autoSelect; therefore, this function returns true.

Constructors

You can construct and destruct objects of the ISettingButton class from derived classes. You cannot copy or assign ISettingButton objects because both the copy constructor and assignment operator are private functions.

ISettingButton

```
ISettingButton();
```

Win	PM	Motif
Y	Y	Y

Because this class is an abstract base class, you cannot construct objects of this class. This constructor is provided for use in derived classes.

~ISettingButton

```
virtual
  ~ISettingButton();
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables the button to send notifications to any observer objects.

```
virtual ISettingButton&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Selection

Use these members to query and change the selection state of a setting button. The selection state of a button is typically changed by the user clicking on it using the mouse or pressing a key to select it.

deselect

Deselects the button.

```
virtual ISettingButton&
  deselect();
```

Win	PM	Motif
Y	Y	Y

isSelected

If the button is selected, true is returned. Otherwise, false is returned.

```
Boolean
  isSelected() const;
```

Win	PM	Motif
Y	Y	Y

select

Selects the button. This function does not call ISelectHandler. Use IButton::click (p. 137) instead to ensure that ISelectHandler is called.

```
virtual ISettingButton&
  select( Boolean select = true );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IButton		
allowsMouseClickFocus	disableMouseClickFocus	highlight
backgroundColor	enableMouseClickFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

© IBM Corp. 1992, 1996

ISettingButton

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

passEventToOwner

Determines if the event is passed on to the owner.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	*N*	*Y*

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

selectId Notification identifier provided to observers when the selection state of a setting button control changes. ISettingButton provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the setting button is selected, and false if the setting button is not selected.

```
static INotificationId const
  selectId;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IButton		
buttonClickId	**noPointerFocus**	

ITextControl		
textId		

IControl		
group	**tabStop**	

IWindow		
activeColorId	**disabledBackgroundColorId**	**noStyle**
backgroundColorId	**disabledForegroundColorId**	**positionId**
borderColorId	**enableId**	**saveBits**
clipChildren	**focusId**	**shadowColorId**
clipSiblings	**fontId**	**sizeId**
clipToParent	**foregroundColorId**	**synchPaint**

© IBM Corp. 1992, 1996

ISettingButton

IWindow		
commandId	**hiliteBackgroundColorId**	**systemCommandId**
deleteId	**hiliteForegroundColorId**	**visible**
disabled	**inactiveColorId**	**visibleId**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ISettingButtonNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          IButtonNotifyHandler
            ISettingButtonNotifyHandler

Inherited By None.

Header File isetbnhd.hpp

Members

Member	Page
Constructor	863
dispatchHandlerEvent	864
~ISettingButtonNotifyHandler	864

The ISettingButtonNotifyHandler class processes events for all classes of setting buttons.

This class is designed to handle events that require the setting button class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ISettingButtonNotifyHandler

This is the default constructor and accepts no parameters.

```
ISettingButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

ISettingButtonNotifyHandler

~ISettingButtonNotifyHandler

```
virtual
  ~ISettingButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the setting button observers if the following event is received:

selected event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IButtonNotifyHandler		
dispatchHandlerEvent		

ITextControlNotifyHandler		
dispatchHandlerEvent		

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ISharedResource

ISharedResource

Derivation

IBase
IVBase
IResource
ISharedResource

Inherited By None.

Header File ireslock.hpp

Members

Member	Page	Member	Page
Constructor	867	keyName	868
handle	868	~ISharedResource	867

The ISharedResource class defines a resource that can be shared across multiple processes. Use the name you provide on construction to distinguish between shared resources.

W32s Resource locks are not supported. Instances of the ISharedResource class can be created and the member functions can be called, but no system resource locks are obtained.

Motif You might use ISharedResource objects to serialize resource access with applications that were not written using the User Interface Class Library. To make this work correctly, your non-User Interface Class Library application needs to use the AIX semaphore object in a manner consistent with the User Interface Class Library. To help you understand how the semaphore object is used by the User Interface Class Library, the basic interaction with the system's subroutines is described below.

The AIX semop subroutine is used for the semaphore lock and release operations. The ISharedResource constructor uses the semget subroutine to allocate a set containing two semaphores. The first semaphore in this set represents the state of the lock. The constructor initializes the semval field to free (value of 0). A lock operation increments the value, while an unlock operation decrements the value.

Recursive lock operations by the same process are allowed. In this case, the resource remains owned by the process until an equal number of unlock operations are done or the object is deleted. The second semaphore in the set is used as a reference counter to indicate the number of ISharedResource objects that are referencing this semaphore set. The constructor increments this counter and the

© Copyright IBM Corp. 1992, 1996

destructor decrements it. When the counter reaches 0, the destructor calls the semctl subroutine to delete the semaphore set.

The constructor determines the semaphore identifier to use in the semget subroutine by using the ftok system subroutine. If the parameter provided on the constructor begins with a slash (/) or is an existing file, the parameter is used as a file name for the ftok call and is the basis of the semaphore identifier token. If the parameter is not the name of an existing file, the constructor interprets the string as the name of a file in the /tmp directory. If this file does not exist, it is created. You can use keyName to obtain the file name used on the ftok subroutine call. The constructor always uses a value of 1 for the second parameter of ftok.

Use a keyName value that is a valid file name in all of the target environments.

Public Functions

Constructors

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

ISharedResource

You can only construct objects of this class by providing a name that uniquely identifies the resource to be shared.

```
ISharedResource( const char* keyName );
```

Win	PM	Motif
Y	Y	Y

keyName Pointer to the name that uniquely identifies the resource.

Resource locks are not supported. No resources are created when calling this constructor.

Exceptions	
IAccessError	The shared resource object was not created. The memory may be exhausted, the handle limit on your platform may be exceeded, or the *keyName* may be invalid.

~ISharedResource

```
virtual
  ~ISharedResource();
```

Win	PM	Motif
Y	Y	Y

Resource locks are not supported. No resources are released when calling this destructor.

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The shared resource object was not deleted. The semaphore handle may be invalid.

Resource Information

Use these members to access shared resource information.

keyName Returns the name used to identify this shared resource.

```
IString
  keyName() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IResource		
lock	unlock	

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Resource Handle

Use these members to obtain a handle to the shared resource.

handle Returns the handle for the shared resource.

```
ISemaphoreHandle&
  handle();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IResource		
handle		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IShowListHandler

Derivation

IBase
IVBase
IHandler
IShowListHandler

Inherited By None.

Header File islhdr.hpp

Members

Member	Page	Member	Page
Constructor	872	listShown	872
dispatchHandlerEvent	872	~IShowListHandler	871

The IShowListHandler-derived classes process IControlEvent events for objects of the class IComboBox (p. 182). For example, a show list event occurs when the user selects the drop-down button of the combination box.

You can create a handler derived from IShowListHandler and attach it to a combination box or to the combination box's owner window. You do this by calling IHandler::handleEventsFor (p. 413) to pass the combination box or owner window to the show list handler.

When the show list handler receives a show list event, it creates an IControlEvent (p. 227) object and routes that object to the IShowListHandler::listShown (p. 872) virtual function. Override this virtual function to supply your own specialized processing of a show list event.

The following return values from the virtual function specifies whether the show list event is passed on for additional processing:

true The show list event requires no additional processing. Do not pass it to another handler.

false The show list event requires additional processing. Pass the show list event to the next handler for additional processing, as follows:

- If there is another handler for the combination box, pass the show list event to the next handler.

© Copyright IBM Corp. 1992, 1996

- If this is the last handler for the combination box, call IWindow::dispatch (p. 1082) to dispatch the show list event to the combination box's owner window.
- For Presentation Manager, if this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the show list event.

Public Functions

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor provided for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

~IShowListHandler

```
virtual
  ~IShowListHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

© IBM Corp. 1992, 1996

IShowListHandler

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor provided for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

IShowListHandler

Used by derived classes to construct objects of this class. This is the default constructor and accepts no parameters.

```
IShowListHandler();
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a show list event is received, the appropriate virtual function is called.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members must be supplied to process a show list event. You can override these virtual members in a derived class.

listShown

Derived classes implement this function to process a show list event.

```
virtual Boolean
  listShown( IControlEvent& event ) = 0;
```

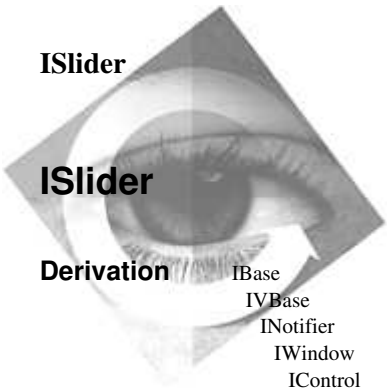
Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ISlider

Derivation

IBase
IVBase
INotifier
IWindow
IControl
IProgressIndicator
ISlider

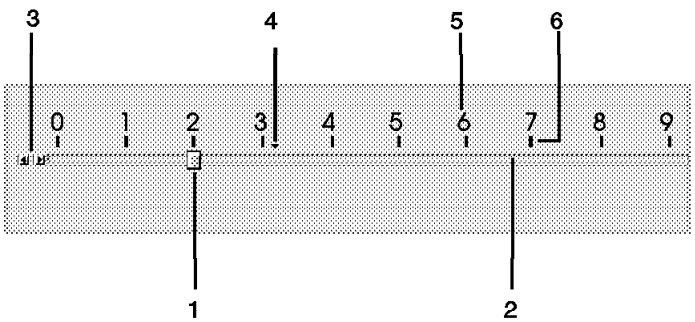
Inherited By None.

Header File islider.hpp

Members

Member	Page	Member	Page
Constructor	878	classDefaultStyle	884
addDetent	880	convertToGUIStyle	881
armSize	876	defaultStyle	881
armTrackId	883	detentPosition	880
buttonsBottom	883	hasFocus	877
buttonsLeft	884	removeDetent	881
buttonsPosition	877	setArmSize	877
buttonsRight	884	setDefaultStyle	881
buttonsTop	884	~ISlider	880

The ISlider class allows a user to set, display, or modify a value by moving the slider arm along the slider shaft.



1 - Slider arm

An arm the user can move along the shaft to set slider values. The slider arm shows the current value by its position on the slider shaft and can be changed by your code, as well as by the user.

© Copyright IBM Corp. 1992, 1996

2 - Slider shaft
A track for the slider arm to move along.

3 - Slider buttons
Buttons that move the slider arm one increment in the indicated direction.

4 - Detent
A user-selectable mark that can be placed anywhere along the slider scale.

5 - Tick text
A label indicating the value that the tick mark represents.

6 - Tick mark
A mark indicating an increment on the slider scale. Tick marks are evenly spaced along the shaft.

Sliders let a user set values that have familiar increments, such as feet or degrees. You can also use sliders to blend colors or show the percentage of a task completed. Because the slider is interactive, detents and buttons are provided and the slider arm is oversized to facilitate interaction with the mouse.

By default, this class creates a slider as follows:

- Horizontally in the center of the control window
- With the slider arm based on the left side
- With the ticks and text above the shaft

The horizontal slider has several alternate styles. You can do the following:

- Place the ticks and text below the shaft
- Place the slider at the top or bottom of the control window
- Base the arm on either the right or left side

A complementary set of options is available for creating a vertical slider. You can do the following:

- Place the ticks and text left or right of the shaft
- Place the slider at the left or right of the control window
- Base the arm on the top or the bottom

Other style options include the following:

- Incremental-movement buttons at either end of the slider
- Filling the slider shaft with color as the arm moves

Handlers derived from the following classes handle events for ISlider objects:

- IEditHandler (p. 267)
- IFocusHandler (p. 322)
- IKeyboardHandler (p. 490)

© IBM Corp. 1992, 1996

- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)
- ISliderArmHandler (p. 888)
- ISliderDrawHandler (p. 892)

Win The native Windows slider (that is, a slider constructed without the pmCompatible (p. 766) style) does not support detents.

Motif The AIX version does not support detents.

The ISlider constructor creates objects of this class using the following Motif widgets:

- An XmDrawingArea widget is created with an XmScale child and one or two XmForm widgets. The IWindow::handle member function returns the handle of the XmDrawingArea widget.
- The XmForm widgets contain the tick marks and tick text.
- The tick marks are XmSeparator widgets and the tick text is implemented with XmLabel widgets. AIX manages the widgets representing ticks when their size is set to a nonzero value.
- XmSeparator widgets are created for each widget during the construction of the ISlider or during processing of the setTicks member function.
- XmLabel widgets are created for the tick text only when setTickText is called.

The User Interface Class Library implements the arrow buttons in an ISlider object by placing two XmArrowButton widgets in an XmForm. The slider behavior is accomplished via arm and disarm callbacks attached to the arrow buttons. Timer functions provide repetitive scrolling in the slider. The snapToTickMark style is implemented by attaching an XmNvalueChanged callback to the scale widget.

Public Functions

Arm Operations

Use these members to set and query attributes of the slider arm.

armSize Returns the size of the arm in pixels.

```
ISize
  armSize() const;
```

Win	PM	Motif
Y	Y	Y

Win Native Windows sliders (that is, sliders constructed without the pmCompatible style) do not support arm sizing. Therefore, this member function returns a default ISize object for native Windows sliders.

setArmSize Sets the size of the arm in pixels.

sliderArm Object that represents the horizontal and vertical dimensions of the slider arm.

```
virtual ISlider&
  setArmSize( const ISize& sliderArm );
```

Win	PM	Motif
Y	Y	I

Win Native Windows sliders (that is, sliders constructed without the pmCompatible style) do not support arm sizing. Therefore, this member function has no effect for native Windows sliders.

Exceptions	
IInvalidParameter	The slider arm size requested is invalid.

Attributes

Use this member to query the accessible attribute of objects of this class.

hasFocus Returns true if the slider has the input focus.

```
virtual Boolean
  hasFocus() const;
```

Win	PM	Motif
Y	N	N

Buttons Position

Use these members to query the position of slider buttons (if any).

buttonsPosition

Returns the current position of the slider's buttons for this slider object. The returned value is an enumerator provided by ButtonsPosition (p. 885).

```
ButtonsPosition
  buttonsPosition() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

© IBM Corp. 1992, 1996

ISlider

1
```
ISlider( unsigned long windowId,
         IWindow* parent,
         IWindow* owner,
         const IRectangle& initial,
         unsigned long scale1NumberOfTicks,
         unsigned long scale1TickSpacing,
         unsigned long scale2NumberOfTicks,
         unsigned long scale2TickSpacing = 0,
         const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	*Y*	*Y*

You can construct an object of this class by specifying the number of ticks and spacing for both scale 1 and scale 2.

windowId A unique ID for the slider control.

parent The parent window.

owner The owner window.

initial A rectangle defining the size and placement of the slider window.

scale1NumberOfTicks
The number of ticks to place on scale 1 of the slider.

scale1TickSpacing
The number of pixels between ticks on scale 1.

scale2NumberOfTicks
The number of ticks to place on scale 2 of the slider.

scale2TickSpacing
The number of pixels between ticks on scale 2. The default is 0. Optional.

style The style of the control. Optional.

Win The constructors wrapping an existing slider are not supported for native Windows sliders.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

2 `ISlider( unsigned long windowId,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial,`
`unsigned long numberOfTicks,`
`unsigned long tickSpacing = 0,`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	Y	Y

You can construct objects of this class by using this primary constructor.

windowId A unique ID for the slider control.

parent The parent window.

owner The owner window.

initial A rectangle defining the size and placement of the slider window.

numberOfTicks
The number of ticks to place on the primary scale of the slider.

tickSpacing
The number of pixels between ticks. The default is 0. The default causes the slider to evenly space the ticks on the shaft. If you specify a value for *tickSpacing*, this constructor determines the length of the slider shaft based on this value and the number of ticks. Optional.

style The style of the control. Optional.

Motif The constructors wrapping an existing progress indicator are not available in the AIX environment.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

3 `ISlider( unsigned long windowId,`
`IWindow* parent );`

Win	PM	Motif
Y	Y	N

You can construct an object of this class by specifying a parent window and a slider control ID.

windowId A unique ID for the slider control.
parent The parent window.

4 `ISlider( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

ISlider

You can construct an object of this class by specifying a window handle.

handle The window handle of the slider control.

Win The constructors wrapping an existing slider are not supported for native Windows sliders.

~ISlider

```
virtual
  ~ISlider();
```

Win	PM	Motif
Y	Y	Y

Detent Operations

Use these members to set and query attributes for a slider's detents.

addDetent

Adds a detent to the slider at the specified pixel offset from the home position. Returns a unique ID, which is required for removing a detent or querying its position.

```
virtual unsigned long
  addDetent( unsigned long offset );
```

Win	PM	Motif
Y	Y	I

Win Native Windows sliders (that is, sliders constructed without the pmCompatible (p. 766) style) do not support detents. Therefore, this member function returns has no effect for native Windows sliders.

Exceptions	
IInvalidParameter	The detent offset requested is invalid.

detentPosition

Returns the offset of a detent from the home position, in pixels.

detentId The detent identifier whose position is to be returned.

```
virtual unsigned long
  detentPosition( unsigned long detentId ) const;
```

Win	PM	Motif
Y	Y	I

Win Native Windows sliders (that is, sliders constructed without the pmCompatible (p. 766) style) do not support detents. Therefore, this member function returns 0 for native Windows sliders.

Exceptions	
IInvalidParameter	The detent identifier is invalid.

removeDetent

Removes the specified detent from the slider.

detentId Detent to remove.

```
virtual ISlider&
  removeDetent( unsigned long detentId );
```

Win	PM	Motif
Y	Y	I

Win Native Windows sliders (that is, sliders constructed without the pmCompatible (p. 766) style) do not support detents. Therefore, this member function has no effect for native Windows sliders.

Exceptions	
IInvalidParameter	The detent identifier is invalid.

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the slider styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IProgressIndicator Styles (p. 763)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 884) unless you have changed the style using setDefaultStyle (p. 881).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent sliders.

© IBM Corp. 1992, 1996

style Use the styles provided by ISlider Styles (p. 883) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IProgressIndicator		
alignment	foregroundColor	setBackgroundColor
armPixelOffset	homePosition	**setDefaultStyle**
armRange	isDrawItemEnabled	setForegroundColor
armTickOffset	isPMCompatible	setHomePosition
backgroundColor	isRibbonStripEnabled	setPrimaryScale
convertToGUIStyle	isSnapToTickEnabled	setShaftBreadth
defaultStyle	isVertical	setShaftPosition
disableDrawItem	moveArmToPixel	setTickLength
disableRibbonStrip	moveArmToTick	setTicks
disableSnapToTick	moveSizeTo	setTickText
enableDrawItem	numberOfTicks	shaftPosition
enableNotification	primaryScale	shaftSize
enableRibbonStrip	resetBackgroundColor	tickLength
enableSnapToTick	resetForegroundColor	tickPosition

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 (p. 1044) are not shown.

Inherited Protected Functions

IProgressIndicator		
calcMinimumSize	initialize	registerCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use this member to identify notifications sent to an observer object.

armTrackId Notification identifier provided to observers when the arm of an ISlider window is dragged with the mouse. ISlider provides the current arm position in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  armTrackId;
```

Win	PM	Motif
Y	*Y*	*Y*

Styles

These style members provide a set of valid styles for this class. Use these members to query and set the slider styles. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IProgressIndicator Styles (p. 763)

buttonsBottom

Places the slider's buttons below the slider shaft.

© IBM Corp. 1992, 1996

```
static const Style
  buttonsBottom;
```

Win	PM	Motif
Y	Y	Y

buttonsLeft Places the slider's buttons to the left of the slider shaft.

```
static const Style
  buttonsLeft;
```

Win	PM	Motif
Y	Y	Y

buttonsRight Places the slider's buttons to the right of the slider shaft.

```
static const Style
  buttonsRight;
```

Win	PM	Motif
Y	Y	Y

buttonsTop Places the slider's buttons above the slider shaft.

```
static const Style
  buttonsTop;
```

Win	PM	Motif
Y	Y	Y

classDefaultStyle

Provides the original default style for this class, which is the following:
IWindow::visible | IProgressIndicator::horizontal | IProgressIndicator::alignCentered | IProgressIndicator::homeLeft | IProgressIndicator::primaryScale1 | ISlider::buttonsLeft.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

IProgressIndicator		
alignBottom	**classDefaultStyle**	**pmCompatible**
alignCentered	**handleDrawItem**	**primaryScale1**
alignLeft	**homeBottom**	**primaryScale2**
alignRight	**homeLeft**	**ribbonStrip**
alignTop	**homeRight**	**scaleId**
armChangeId	**homeTop**	**snapToTickMark**
border3D	**horizontal**	**vertical**

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ISlider contains the following nested classes:

ISlider::Style (see page 887)

ButtonsPosition

```
ButtonsPosition {
    top,    bottom, left,   right,  none
    };
```

Use these enumerators to specify the placement of the slider's buttons. The buttons are used to move the slider arm one increment in the selected direction.

top
Places the slider's buttons above the slider shaft.

bottom
Places the slider's buttons below the slider shaft.

left
Places the slider's buttons to the left of the slider shaft.

right
Places the slider's buttons to the right of the slider shaft.

© IBM Corp. 1992, 1996

none
The slider has no buttons.

ISlider::Style

Derivation IBase
IBitFlag
ISlider::Style

Inherited By None.

Header File islider.hpp

The nested class ISlider::Style provides a set of valid styles for the ISlider (p. 874) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

ISliderArmHandler

Derivation

IBase
IVBase
IHandler
ISliderArmHandler

Inherited By None.

Header File isldahdr.hpp

Members

Member	Page	Member	Page
Constructor	890	positionChanged	890
dispatchHandlerEvent	890	~ISliderArmHandler	889
moving	890		

The ISliderArmHandler class handles events that result from movement of the slider arm, either by the user moving the slider arm or by calling one of the functions that change the slider arm position. You can use this handler for both ISlider (p. 874) and ICircularSlider (Vol. III) controls.

You create a handler derived from ISliderArmHandler and attach it to either the slider control or to the control's owner window. You can do this by calling IHandler::handleEventsFor (p. 413) to pass the appropriate slider control window or owner window to the slider arm handler.

Slider movement events get generated when the user moves the arm of the slider. When the slider arm handler receives a slider movement event, it creates an IControlEvent (p. 227) object and routes that object to the ISliderArmHandler::moving (p. 890) virtual function. You can override this virtual function to supply your own specialized processing of a slider movement event.

Position change events get generated when the user changes the sliders' position using the increment or decrement buttons or calls a function that moves the slider. When the slider arm handler receives a position change event, it creates an IControlEvent (p. 227) object and routes that object to the ISliderArmHandler::positionChanged (p. 890) virtual function. You can override this virtual function to supply your own specialized processing of a position change event.

The following return values from the virtual functions specify whether the control event is passed on for additional processing:

© Copyright IBM Corp. 1992, 1996

true The movement or position change event requires no additional processing. Do not pass it to another handler.

false The movement or position change event requires additional processing. Pass the movement or position change event to the next handler for additional processing, as follows:

- If there is another handler for the control, pass the movement or position change event to the next handler.
- If this is the last handler for the control, call IWindow::dispatch (p. 1082) to dispatch the event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the event.

Public Functions

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

~ISliderArmHandler

```
virtual
  ~ISliderArmHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

Protected Functions

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

ISliderArmHandler

Used by derived classes to construct objects of this class. This is the default constructor and accepts no parameters.

```
ISliderArmHandler();
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a movement or position change event is received, the appropriate virtual function is called.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members must be supplied to process control events for slider arm movements and for changes in position. You can override these virtual members in a derived class.

moving

Processes slider arm movement events. It is implemented by derived classes.

```
virtual Boolean
  moving( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

positionChanged

Processes position change events. It is implemented by derived classes.

```
virtual Boolean
  positionChanged( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ISliderDrawHandler

Derivation

IBase
IVBase
IHandler
ISliderDrawHandler

Inherited By None.

Header File islidhdr.hpp

Members

Member	Page	Member	Page
Constructor	893	drawRibbonStrip	895
dispatchHandlerEvent	894	drawShaft	895
drawArm	894	~ISliderDrawHandler	893
drawBackground	895		

The ISliderDrawHandler class draws slider and progress indicator control objects. You can draw the following:

- Slider shaft
- Ribbon strip
- Slider arm
- Slider background

To use this handler, you must create the slider or progress indicator with the handleDrawItem style specified, or you must set this style by using IProgressIndicator::enableDrawItem after the slider or progress indicator is created. See IProgressIndicator Styles (p. 763) for information about the handleDrawItem style and enableDrawItem (p. 751) for information about that function.

Create a handler derived from ISliderDrawHandler and attach it to a slider or progress indicator. You can do this by calling IHandler::handleEventsFor to pass the slider or progress indicator to the slider draw handler. See handleEventsFor (p. 413) for information about that function.

When the slider draw handler receives a slider draw event, it creates an IDrawItemEvent object and routes that object to the appropriate ISliderDrawHandler virtual function. Override these virtual functions to supply your own specialized processing of a slider draw event. See IDrawItemEvent (p. 258) for information about that class.

© Copyright IBM Corp. 1992, 1996

The return value from the virtual functions specifies whether the slider draw event should be passed on for additional processing, as follows:

Return Value	Meaning
true	The slider draw event requires no additional processing. Do not pass it to another handler.
false	The slider draw event requires additional processing. Pass the slider draw event to the next handler for additional processing, as follows: • If there is another handler for the slider or progress indicator, pass the slider draw event to the next handler. • If this is the last handler for the slider or progress indicator, call the IWindow::dispatch function to dispatch the slider draw event to the slider's or progress indicator's owner window. See dispatch (p. 1082) for information about that function. • If this is the last handler for the owner window, call the IWindow::defaultProcedure function to process the slider draw event. See defaultProcedure (p. 1082) for information about that function.

Public Functions

Constructors

You can construct and destruct objects of this class.

ISliderDrawHandler

Used to construct objects of this class. This is the default constructor and accepts no parameters.

```
ISliderDrawHandler();
```

Win	PM	Motif
Y	Y	Y

~ISliderDrawHandler

```
virtual
  ~ISliderDrawHandler();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

ISliderDrawHandler

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. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a slider draw event is received, the appropriate virtual function is called.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members must be supplied to process slider drawing events You can override these virtual members in a derived class.

drawArm Draws the slider's arm. The application overrides this function to handle the drawing.

```
virtual Boolean
  drawArm( IDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

drawBackground

Draws the slider's background. The application overrides this function to handle the drawing.

```
virtual Boolean
  drawBackground( IDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

drawRibbonStrip

Draws the slider's ribbon strip. The application overrides this function to handle the drawing.

```
virtual Boolean
  drawRibbonStrip( IDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

drawShaft

Draws the slider's shaft. The application overrides this function to handle the drawing.

```
virtual Boolean
  drawShaft( IDrawItemEvent& event );
```

Win	PM	Motif
Y	Y	Y

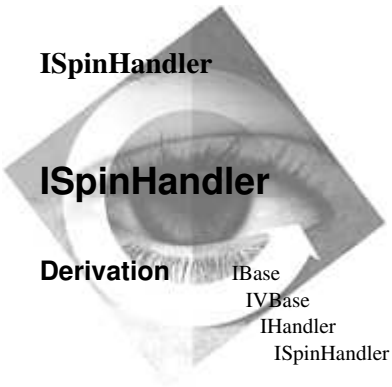
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



ISpinHandler

Derivation

IBase
IVBase
IHandler
ISpinHandler

Inherited By None.

Header File ispinhdr.hpp

Members

Member	Page	Member	Page
Constructor	898	dispatchHandlerEvent	898
arrowDown	898	spinEnded	899
arrowUp	898	~ISpinHandler	897

The ISpinHandler class processes spin events for objects of the following classes:

- INumericSpinButton (p. 672)
- ITextSpinButton (p. 963)
- ISpinButton (obsolete)

ISpinHandler handles events resulting from a user interacting with a spin button control, such as when the user selects the down arrow in a spin button. When the spin handler receives a spin event, it creates an IControlEvent (p. 227) object and routes that object to the appropriate ISpinHandler virtual function.

You can create a handler derived from ISpinHandler and attach it to a spin button or to the spin button's owner window by calling IHandler::handleEventsFor (p. 413) to pass the spin button or owner window to the spin handler. By overriding the virtual functions supplied by ISpinHandler, you can provide customized processing of a spin event.

The return value from the virtual functions specifies whether the spin event is passed on for additional processing, as follows:

true The spin event requires no additional processing. Do not pass it to another handler.

false The spin event requires additional processing. Pass the spin event to the next handler for additional processing, as follows:

- If there is another handler for the spin button, pass the spin event to the next handler.

© Copyright IBM Corp. 1992, 1996

- If this is the last handler for the spin button, call IWindow::dispatch (p. 1082) to dispatch the spin event to the spin button's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (p. 1082) to process the spin event.

  Note: There is no defaultProcedure in the AIX environment.

Public Functions

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

~ISpinHandler

```
virtual
  ~ISpinHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

© IBM Corp. 1992, 1996

ISpinHandler

Constructors

The only way to create objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. This default constructor can be used by derived classes to create objects of this class. You can destruct objects of this class.

ISpinHandler Used by derived classes to construct objects of this class. This is the default constructor and accepts no parameters.

```
ISpinHandler();
```

Win	PM	Motif
Y	Y	Y

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process it. If it is, it calls the virtual function used to process the event.

dispatchHandlerEvent

If a spin event is received, the appropriate virtual function is called.

event An IEvent object constructed as a result of a user or library action.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Event Processing

Event-processing members must be supplied to process a spin event. You can override these virtual members in a derived class.

arrowDown Implemented by derived classes to process the pressing of the down arrow key or clicking on the down arrow of the spin button.

event An IControlEvent object constructed as a result of a user action.

```
virtual Boolean
  arrowDown( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

arrowUp Implemented by derived classes to process the pressing of the up arrow key or clicking on the up arrow of the spin button.

event An IControlEvent object constructed as a result of a user action.

```
virtual Boolean
  arrowUp( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

spinEnded Implemented by derived classes to process the release of an arrow key or the mouse button while spinning the spin button.

event An IControlEvent object constructed as a result of a user action.

```
virtual Boolean
  spinEnded( IControlEvent& event );
```

Win	PM	Motif
Y	Y	Y

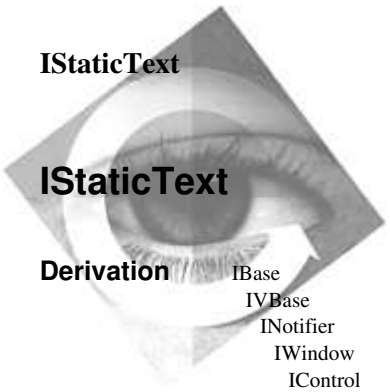
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IStaticText

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IStaticText

Inherited By

IBitmapControl
IInfoArea

Header File istattxt.hpp

Members

Member	Page	Member	Page
Constructor	903	hasFillBackground	905
alignment	901	isHalftone	908
backgroundColor	902	isStrikeout	908
border3D	912	isUnderscore	908
bottom	912	left	912
calcLimitSize	910	limit	905
calcMinimumSize	910	limitId	911
center	912	mnemonic	913
classDefaultStyle	912	moveSizeTo	908
convertToGUIStyle	906	passEventToOwner	909
defaultStyle	906	resetFillColor	902
disableFillBackground	904	right	913
disableHalftone	907	setAlignment	901
disableStrikeout	907	setDefaultStyle	906
disableUnderscore	907	setFillColor	903
enableFillBackground	904	setLimit	905
enableHalftone	907	setText	908
enableStrikeout	907	strikeout	913
enableUnderscore	907	strikeoutId	911
fillBackground	912	top	913
fillBackgroundId	910	underscore	913
fillColor	902	underscoreId	911
fillColorId	911	vertCenter	913
foregroundColor	902	wordBreak	913
halftone	912	~IStaticText	904
halftoneId	911		

The IStaticText class creates and manages static text control windows. *Static text controls* are simple text fields that do not accept user input. For example,

© Copyright IBM Corp. 1992, 1996

```
IStaticText sttxtText(ID_TEXT, this, this, IRectangle(10,10,70,24)));
sttxtText.setText("It is a wonderful day");
```

You can use the classes derived from IStaticText to display bitmaps and icons.

You can use IStaticText to create text prompts and labels. You can give the static text a background color so that it has the appearance of a box. By using a background color and no text, you can create a solid box with nothing inside it.

You can attach the following handlers to this control:

- IMouseHandler (p. 631)
- IPaintHandler (p. 706)
- IResizeHandler (p. 802)

PM If you do not set a fill color, the background color is used as the fill color.

OS/2 applications can use the IStaticText control to create boxes that label, box, or frame other controls.

The IStaticText control can be parent, child, or owner of these other controls. However, for portable applications, you should use IFrameWindow, IOutlineBox, or the canvas classes instead of IStaticText.

Public Functions

Alignment

Use these members to query and set the alignment of the static text control.

alignment Returns the current alignment for this static text object. The returned value is an Alignment (p. 915) enumerator.

```
Alignment
  alignment() const;
```

Win	PM	Motif
Y	Y	Y

setAlignment Sets the alignment of the static text.

```
virtual IStaticText&
  setAlignment( Alignment alignment = IStaticText::topLeft );
```

Win	PM	Motif
Y	Y	Y

alignment Use the enumerators provided by Alignment (p. 915) to specify the alignment of the static text.

© IBM Corp. 1992, 1996

Exceptions	
IInvalidParameter	The alignment value specified is invalid. See the documentation for the Alignment group under IStaticText for valid alignment values.

Colors

Use these members to query, set, and reset colors for the static text control. You can query the foreground and background colors. You can query and set the fill color for the static text.

backgroundColor

Returns the background color of the static text object. If you do not set a color for this area, the default is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	Y	N

PM The default is from the owner window chain or color scheme on the OS/2 platform.

This member is overridden in this derived class for specific operating system behavior.

fillColor

Returns the fill color value of the static text. If you do not set the fill color, the default is returned.

```
virtual IColor
  fillColor() const;
```

Win	PM	Motif
Y	Y	Y

Motif Fill color and background color are identical in the AIX environment. Therefore, this function returns the background color in this environment.

foregroundColor

Returns the foreground color value of the static text. If you do not set the color for the area, the default is returned.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	Y	N

This member is overridden in this derived class for specific operating system behavior.

resetFillColor

Resets the fill color by undoing a previous set.

```
virtual IStaticText&
  resetFillColor();
```

Win	PM	Motif
Y	Y	Y

Motif This function has no effect in the AIX environment.

setFillColor Sets the color for the area of the static text control window that does not have text.

```
virtual IStaticText&
  setFillColor( const IColor& color );
```

Win	PM	Motif
Y	Y	Y

Motif This function is equivalent to setBackgroundColor in the AIX environment.

Constructors

You can construct and destruct objects of the IStaticText class. You cannot copy or assign IStaticText objects because both the copy and assignment operator are private functions. You can construct objects of this class in the following ways:

- Create the specified static text control and an object for it
- Create the object for the specified static text control

IStaticText

1

```
IStaticText( unsigned long id,
             IWindow* parent,
             IWindow* owner,
             const IRectangle& initial = IRectangle ( ),
             const Style& style = defaultStyle ( ) );
```

Win	PM	Motif
Y	Y	Y

Creates a static text control and an object for it.

id A static text control ID.

parent The parent window.

owner The owner window.

initial The initial position and size of the control you are constructing. Optional.

style The control's characteristics. Optional.

Exceptions	
IInvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.

© IBM Corp. 1992, 1996

IStaticText

2 `IStaticText( unsigned long id, IWindow* parent );`

Win	PM	Motif
Y	Y	Y

Creates an object for a static text control that exists in a dialog window.

id The ID of the existing static text control.
parent The parent window.

3 `IStaticText( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	Y

Creates an object for an existing static text control.

handle The window handle of an existing static text control.

~IStaticText

```
virtual
  ~IStaticText();
```

Win	PM	Motif
Y	Y	Y

Fill Background

Use these members to query and set the fill background of the static text control. If fill background is enabled, the background of the static text control is erased before the text is drawn.

disableFillBackground

Draws text over the current background.

```
virtual IStaticText&
  disableFillBackground();
```

Win	PM	Motif
Y	Y	Y

Motif The fillBackground style is always enabled in AIX; therefore, this function has no effect.

enableFillBackground

Erases the background before the text is drawn.

Note: If you call this function on an object of a derived class, such as IBitmapControl or IIconControl, the graphic is overlaid by the background color.

```
virtual IStaticText&
  enableFillBackground( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

enable The default erases the background before the text is drawn. If you specify false, the background is not changed when the text is drawn. Therefore, when enable is false, this function acts like disableFillBackground (p. 904).

Motif The fillBackground style is ignored in AIX; therefore, this function has no effect.

hasFillBackground

If the background is erased before any text is drawn, true is returned. Otherwise, false is returned.

```
Boolean
  hasFillBackground() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The fillBackground style is ignored in AIX. This function returns false.

Minimum Size Adjustment

Use these members to set and query the number of characters that the minimum size can be based on.

limit

Returns the number of characters set by setLimit. You can use this limit or calcMinimumSize to determine a minimum limit for the static text control.

```
unsigned long
  limit() const;
```

Win	PM	Motif
Y	*Y*	*Y*

setLimit

Sets the minimum number of characters in a maximum size static text control.

Use this for static text controls that change the text displayed on an ISetCanvas (Vol. III) or IMultiCellCanvas (Vol. III) to avoid causing the canvas to update the layout of its child windows with each change to the static text contents.

calcLimitSize (p. 910) uses a nonzero limit value to calculate a minimum size for this window. The size can be larger than the actual size needed to hold the currently displayed text.

The minimum size used is the *larger* of either of the following:

- The calculated minimum size
- The size that accommodates the displayed text

If this function is not called, the minimum size needed to display the current text is the default.

© IBM Corp. 1992, 1996

```
virtual IStaticText&
  setLimit( unsigned long limit = 0 );
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the IStaticText (p. 900) class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle

Returns the default style. The default style is classDefaultStyle (p. 912) unless you have changed it using setDefaultStyle (p. 906).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

setDefaultStyle

Sets the default style for all subsequent static text controls.

style Use the styles provided by IStaticText Styles (p. 910) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Text Processing

Use these members to set the text, to query the text, and to modify how the text is displayed for the static text control.

disableHalftone

Removes the halftone (p. 912) from the text.

```
virtual IStaticText&
  disableHalftone();
```

Win	PM	Motif
Y	Y	Y

Motif The halftone style is not available in AIX; therefore, this function has no effect.

disableStrikeout

Removes the line drawn through the text.

```
virtual IStaticText&
  disableStrikeout();
```

Win	PM	Motif
Y	Y	Y

Motif The strikeout style is not available in AIX; therefore, this function has no effect.

disableUnderscore

Removes the line beneath the text.

```
virtual IStaticText&
  disableUnderscore();
```

Win	PM	Motif
Y	Y	Y

Motif The underscore style is not available in AIX; therefore, this function has no effect.

enableHalftone

Draws the text using the style halftone (p. 912).

```
virtual IStaticText&
  enableHalftone( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Motif The halftone style is not available in AIX; therefore, this function has no effect.

enableStrikeout

Draws a line through the text.

```
virtual IStaticText&
  enableStrikeout( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Motif The strikeout style is not available in AIX; therefore, this function has no effect.

enableUnderscore

Draws a line beneath the text.

```
virtual IStaticText&
  enableUnderscore( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Motif The underscore style is not available in AIX; therefore, this function has no effect.

isHalftone If the text has the style halftone (p. 912) set, true is returned. Otherwise, false is returned.

```
Boolean
  isHalftone() const;
```

Win	PM	Motif
Y	Y	Y

Motif The halftone style is not available in AIX. This function returns false.

isStrikeout If the text has the style strikeout (p. 913) set, true is returned. Otherwise, false is returned.

```
Boolean
  isStrikeout() const;
```

Win	PM	Motif
Y	Y	Y

Motif The strikeout style is not available in AIX. This function returns false.

isUnderscore If the text has the style underscore (p. 913) set, true is returned. Otherwise, false is returned.

```
Boolean
  isUnderscore() const;
```

Win	PM	Motif
Y	Y	Y

Motif The underscore style is not available in AIX. This function returns false.

setText Sets the text for the control. Overrides the inherited setText. If appropriate, this function also notifies a parent canvas to update the layout for its children.

1
```
virtual IStaticText&
  setText( const char* text );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the text in the static text control using a character string.

2
```
virtual IStaticText&
  setText( const IResourceId& text );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the text in the static text control using an IResourceId.

Window Positioning

Use these members to set the size and position of static control windows.

moveSizeTo Changes the position and size of the static text window using a lower-left origin-based coordinate system.

```
virtual IStaticText&
  moveSizeTo( const IRectangle& aRectangle );
```

Win	PM	Motif
N	*N*	*Y*

Motif This member function overrides the IWindow version for Motif. It obtains the proper vertical alignment for static text.

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

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 (p. 1044) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

passEventToOwner

Determines if the event is passed on to the owner.

© IBM Corp. 1992, 1996

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	N	Y

Layout Support

Layout support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

calcLimitSize Returns a size based on the text limit value and the current font.

```
virtual ISize
  calcLimitSize() const;
```

Win	PM	Motif
Y	Y	Y

calcMinimumSize

Returns the recommended minimum size for this static text control. The size is based on the following:

- The text string length
- The current font
- The value returned by IStaticText::calcLimitSize (p. 910)

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

fillBackgroundId

Notification identifier provided to observers when the fill background style of a static text control changes. IStaticText provides a Boolean value in the

INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if fillBackground is now enabled, and false if fillBackground is disabled.

```
static INotificationId const
  fillBackgroundId;
```

Win	PM	Motif
Y	Y	Y

fillColorId Notification identifier provided to observers when the fill color of a static text control changes.

```
static INotificationId const
  fillColorId;
```

Win	PM	Motif
Y	Y	Y

halftoneId Notification identifier provided to observers when the halftone style of a static text control changes. IStaticText provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if halftone is now enabled, and false if halftone is disabled.

```
static INotificationId const
  halftoneId;
```

Win	PM	Motif
Y	Y	Y

limitId Notification identifier provided to observers when the text limit of a static text control changes. IStaticText provides the new limit value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  limitId;
```

Win	PM	Motif
Y	Y	Y

strikeoutId Notification identifier provided to observers when the strikeout style of a static text control changes. IStaticText provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if strikeout is now enabled, and false if strikeout is disabled.

```
static INotificationId const
  strikeoutId;
```

Win	PM	Motif
Y	Y	Y

underscoreId Notification identifier provided to observers when the underscore style of a static text control changes. IStaticText provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value will be true if underscore is now enabled, and false if underscore is disabled.

```
static INotificationId const
  underscoreId;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Styles

These style members provide a set of valid styles for the IStaticText (p. 900) class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)

border3D Adds an etched 3D border to the control.

```
static const Style
  border3D;
```

Win	PM	Motif
Y	I	I

Win This style is ignored on Windows NT 3.51 with Program Manager.

bottom Aligns the text at the bottom of the window

```
static const Style
  bottom;
```

Win	PM	Motif
Y	Y	Y

center Centers the text.

```
static const Style
  center;
```

Win	PM	Motif
Y	Y	Y

classDefaultStyle

Provides the original default style for this class, which is the following: IStaticText::left | IStaticText::top | IStaticText::fillBackground | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	Y	Y

fillBackground

Erases the background, using the currently set fill color, before drawing the text.

```
static const Style
  fillBackground;
```

Win	PM	Motif
Y	Y	Y

halftone Draws the text in halftone color.

```
static const Style
  halftone;
```

Win	PM	Motif
Y	Y	Y

left Left-justifies the text.

```
static const Style
  left;
```

Win	PM	Motif
Y	Y	Y

mnemonic If a mnemonic prefix character (the tilde) is found, the character following the prefix character is drawn with mnemonic emphasis. Without the mnemonic style, the tilde is displayed in the text of the IStaticText object.

```
static const Style
  mnemonic;
```

Win	PM	Motif
Y	Y	Y

right Right-justifies the text.

```
static const Style
  right;
```

Win	PM	Motif
Y	Y	Y

strikeout Draws the text with a line through it.

```
static const Style
  strikeout;
```

Win	PM	Motif
Y	Y	Y

top Aligns the text at the top of the window.

```
static const Style
  top;
```

Win	PM	Motif
Y	Y	Y

underscore Underlines the text.

```
static const Style
  underscore;
```

Win	PM	Motif
Y	Y	Y

vertCenter Aligns the text vertically in the center of the window.

```
static const Style
  vertCenter;
```

Win	PM	Motif
Y	Y	Y

wordBreak Word-wraps, at ends of lines, text that has multiple lines. You can only use this style when you also specify the left and top styles.

```
static const Style
  wordBreak;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

IStaticText

PM In OS/2 Presentation Manager, when you use static text objects in a multicell canvas and your static text requires multiple lines, you must override calcMinimumSize to return a rectangle for more than one line of text.

Motif The User Interface Class Library creates objects of the IStaticText class using XmLabel widgets.

Although AIX does not support the style wordBreak, the XmLabel widget supports multiline text by explicit use of new-line characters.

Inherited Public Data

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IStaticText contains the following nested classes:

IStaticText::Style (see page 916)

Alignment

```
Alignment {
    topLeft,        topLeftWrapped, topCenter,      topRight,       centerLeft,
    centerCenter,   centerRight,    bottomLeft,     bottomCenter,   bottomRight
    };
```

Use these enumerators to specify the alignment of the static text:

topLeft
Aligns the text at the top of the window and left-justifies the text.

topLeftWrapped
Aligns the text at the top of the window, left-justifies the text, and wraps the text into multiple lines using word-wrapping at the end of the lines.

topCenter
Aligns text at the top of the window and centers it.

topRight
Aligns text at the top of the window and right-justifies it.

centerLeft
Aligns text vertically in the center of the window and left-justifies it.

centerCenter
Aligns text vertically in the center of the window and centers it.

centerRight
Aligns text vertically in the center of the window and right-justifies it.

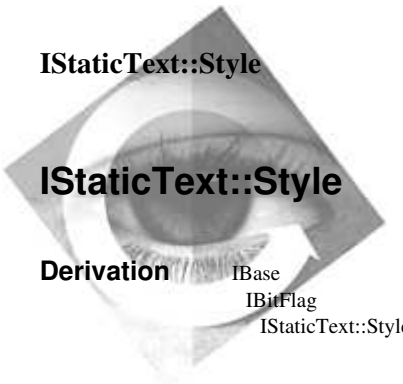
bottomLeft
Aligns text at the bottom of the window and left-justifies it.

bottomCenter
Aligns text at the bottom of the window and centers it.

bottomRight
Aligns text at the bottom of the window and right-justifies it.

Motif In AIX, topLeftWrapped is equivalent to the enumerator topLeft. Use topLeft alignment whenever the static text contains new-line separators.

© IBM Corp. 1992, 1996



IStaticText::Style

Derivation

IBase
IBitFlag
IStaticText::Style

Inherited By None.

Header File istattxt.hpp

The nested class IStaticText::Style provides a set of valid styles for the IStaticText (p. 900) class.

AIX does not support several styles. You can use these styles for portable applications, but AIX will ignore these styles.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996

IStringHandle

Derivation

IBase
  IHandle
    IStringHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	917
operator Value	917

The IStringHandle class accesses a string.

PM IStringHandle is an alias for the OS/2 Programmer's Toolkit type HSTR.

Public Functions

Constructors

You can construct objects of this class.

IStringHandle

You can construct objects of this class from a string handle (a value of type IHandle::Value), which defaults to 0.

```
IStringHandle( Value hstr = 0 );
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

© Copyright IBM Corp. 1992, 1996

IStringHandle

```
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**	

ISubmenu

Derivation

IBase
IVBase
INotifier
IWindow
IMenu
ISubmenu

Inherited By None.

Header File isubmenu.hpp

Members

Member	Page	Member	Page
Constructor	923	elementAt	924
add	924	enableItem	927
addAsNext	924	handle	925
addBitmap	920	menuHandle	925
addItem	921	removeSubmenu	925
addSeparator	922	removeSubmenuAt	924
addText	922	setBitmap	926
checkItem	927	setItem	927
cursor	924	setText	927
deleteAt	924	uncheckItem	928
deleteItem	925	undo	928
disableItem	927	~ISubmenu	923

The ISubmenu class creates a submenu object from a menu handle. This class overrides functions of IMenu (p. 525) to provide menu item functions that can be restored. The menu item functions in this class act upon this ISubmenu object. Any changes are undone when the menu ends.

Motif The User Interface Class Library adds a Motif callback routine to all submenus to enable processing of the following menu events:

- Showing a menu on the display
- Selecting a menu item in a menu
- Removing a menu from the display

Public Functions

© Copyright IBM Corp. 1992, 1996

Adding Items

You can add textual, graphical, submenu, or separator items to a submenu object. You can also construct an object of the IMenuItem (p. 582) class and add it to the submenu object. In these overridden versions, the item is always added to this ISubmenu object. You do not need to specify the *intoSubmenuId* parameter because it is always assumed to have the value of the identifier of this submenu.

Note: Use a value between 1 and 65565 for the item identifier in these functions. Values outside the recommended range can be truncated by the underlying GUI or result in portability problems. In addition, use a unique value for each menu item within a particular menu or submenu. You need unique menu item identifiers if you want to access items using this class and to identify events resulting from user selection of menu items.

addBitmap Adds a bitmap menu item as the last item in a menu or submenu.

1

```
virtual ISubmenu&
  addBitmap( unsigned long newItemId,
             const IResourceId& newItemBitmapResId,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to load a bitmap from the resource library that you specified when you created the IResourceId object. The parameters are the following:

newItemId The identifier of the new menu item to add to a menu or submenu.

newItemBitmapResId
The resource identifier of the bitmap for the new menu item.

intoSubmenuID
Not used. It is only present to allow this function to override the function inherited from IMenu.

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

2

```
virtual ISubmenu&
  addBitmap( unsigned long newItemId,
             const IBitmapHandle& itemBitmap,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to add a bitmap you already have loaded into a menu item. The parameters are the following:

newItemId The identifier of the new menu item to add to a menu or submenu.

itemBitmap
The handle of the bitmap for the new menu item.

intoSubmenuID
Not used. It is only present to allow this function to override the function inherited from IMenu.

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

3
```
virtual ISubmenu&
  addBitmap( unsigned long newItemId,
             unsigned long newItemBitmapResId,
             unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this version to load a bitmap from the default resource library. The parameters are the following:

newItemId The identifier of the new menu item to add to a menu or submenu.

newItemBitmapResId
The identifier of the bitmap for the new menu item.

intoSubmenuID
Not used. It is only present to allow this function to override the function inherited from IMenu.

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

addItem Adds a menu item represented by an IMenuItem (p. 582) object as the last item in a menu or submenu. The optional *intoSubmenuId* is ignored. It is only present to allow this function to override the function inherited from IMenu.

© IBM Corp. 1992, 1996

ISubmenu

```
virtual ISubmenu&
  addItem( IMenuItem& menuItem,
           unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

addSeparator

Adds a separator menu item with the specified identifier as the last item in a submenu.

1
```
virtual ISubmenu&
  addSeparator( unsigned long newItemId = 0 );
```

Win	PM	Motif
Y	Y	Y

Note: IMenu also has an addSeparator function that accepts a single parameter. But while the parameter for IMenu::addSeparator (p. 529) is *intoSubmenuId*, the parameter for this version of addSeparator is *newItemId*.

2
```
virtual ISubmenu&
  addSeparator( unsigned long newItemId,
                unsigned long intoSubmenuId );
```

Win	PM	Motif
Y	Y	Y

addText

Adds a text menu item as the last item in a menu or submenu. The optional *intoSubmenuId* is ignored. It is only present to allow this function to override the function inherited from IMenu.

1
```
virtual ISubmenu&
  addText( unsigned long newItemId,
           const IResourceId& newItemTextResId,
           unsigned long intoSubmenuId = 0 );
```

Win	PM	Motif
Y	Y	Y

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

2 `virtual ISubmenu&`
`  addText( unsigned long newItemId,`
`           const char* itemText,`
`           unsigned long intoSubmenuId = 0 );`

Win	PM	Motif
Y	*Y*	*Y*

Motif X-Motif provides specific widgets and gadgets to use for specific purposes in menus. Pull-down menus can contain the following widgets and gadgets:

- CascadeButton
- ToggleButton
- Label
- Separator

Constructors

You can construct and destruct objects of this class. You cannot copy or assign ISubmenu objects because both the copy constructor and assignment operator are private functions.

ISubmenu

Constructs an ISubmenu object from the handle of a menu. Typically, only menu handlers create ISubmenu objects.

1 `ISubmenu( const IWindowHandle& submenuHandle );`

Win	PM	Motif
N	*N*	*Y*

Creates an ISubmenu object from the window handle of a submenu window. One of the primary uses of this constructor is in the implementation of the IMenuHandler (p. 576) class.

2 `ISubmenu( const IMenuHandle& submenuHandle );`

Win	PM	Motif
Y	*Y*	*N*

Creates an ISubmenu object from the handle of a submenu window. One of the primary uses of this constructor is in the implementation of the IMenuHandler (p. 576) class.

~ISubmenu

`virtual`
`  ~ISubmenu();`

Win	PM	Motif
Y	*Y*	*Y*

Cursored Operations

Cursored operations use objects of the ISubmenu::Cursor (p. 931) nested class to designate which menu item is to be acted upon. You can use these members to access each item in the menu without knowing the menu item identifiers. In these overridden versions, information is stored about the change so that the change can be undone.

© IBM Corp. 1992, 1996

ISubmenu

add

Adds the specified menu item at the position of the specified cursor by pushing down everything after the cursor.

```
virtual ISubmenu&
  add( IMenuItem& menuItem,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

addAsNext

Adds the specified menu item as the next item in the submenu and sets the specified cursor there.

```
virtual ISubmenu&
  addAsNext( IMenuItem& menuItem,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

cursor

Returns an ISubmenu::Cursor (p. 931) object and sets the cursor to point to the specified menu item.

```
Cursor
  cursor( unsigned long itemId ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

deleteAt

Deletes the menu item at the position of the specified cursor and sets the cursor to the menu item that precedes the deleted menu item.

```
virtual ISubmenu&
  deleteAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

elementAt

Returns the menu item at the position of the specified cursor.

```
virtual IMenuItem
  elementAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

removeSubmenuAt

Removes the menu item at the cursor position submenu indicator and destroys the submenu. Typically, you use this member function to remove cascading menus.

```
virtual ISubmenu&
  removeSubmenuAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Properties

Use property members to obtain general information about the submenu.

handle Returns the submenu window handle.

```
virtual IWindowHandle
  handle() const;
```

Win	PM	Motif
Y	Y	Y

PM This function is equivalent to menuHandle (p. 925), but menuHandle should be preferred over this function because it will provide better portability in the future.

menuHandle Returns the handle to the submenu.

```
virtual IMenuHandle
  menuHandle() const;
```

Win	PM	Motif
Y	Y	N

Removing Items

These members delete menu items and remove a submenu of the instances of this submenu class. In these overridden versions, information is stored about the change so that the change can be undone.

deleteItem Removes the specified item from a menu.

```
virtual ISubmenu&
  deleteItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu item identifier is invalid. Verify that the identifier represents an item in the menu.

removeSubmenu

Removes the menu item submenu indicator and destroys the submenu. Typically, you use this member function to remove cascading menus.

```
virtual ISubmenu&
  removeSubmenu( unsigned long itemWithSubmenuId );
```

Win	PM	Motif
Y	Y	Y

Replacing Items

You replace a menu item to change the text or graphic displayed by the item or to create a submenu. You can also replace a menu item with an IMenuItem (p. 582) object. You can create an IMenuItem object using the menuItem (p. 539) function, make the appropriate changes

© IBM Corp. 1992, 1996

to the IMenuItem object, and then replace the item in the menu object using the setItem (p. 927) function.

This class overrides inherited functions for replacing menu items. In these overridden versions, information is stored about the change so that the change can be undone. In addition, with these versions, the item is always added to this ISubmenu object. You do not need to specify the *intoSubmenuId* parameter because it is always assumed to have the value of the identifier of this submenu.

setBitmap Replaces a specified menu item with a bitmap.

1
```
virtual ISubmenu&
  setBitmap( unsigned long menuItemId,
             unsigned long newBitmapResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want User Interface Class Library to load the bitmap from the default resource library.

menuItemId
The identifier of the menu item to add the bitmap to.

newBitmapResId
The bitmap to display. This parameter accepts the resource identifier of a bitmap in the default resource library.

2
```
virtual ISubmenu&
  setBitmap( unsigned long menuItemId,
             const IBitmapHandle& bitmapHandle );
```

Win	PM	Motif
Y	Y	Y

Use this function when you have the bitmap loaded.

menuItemId
The identifier of the menu item to add the bitmap to.

bitmapHandle
The handle of the bitmap to display.

3
```
virtual ISubmenu&
  setBitmap( unsigned long menuItemId,
             const IResourceId& newBitmapResId );
```

Win	PM	Motif
Y	Y	Y

Use this function when you want User Interface Class Library to load the bitmap from the resource library you specified when you created the IResourceId object.

menuItemId
The identifier of the menu item to add the bitmap to.

newBitmapResId
The bitmap to display. This parameter accepts the resource identifier of a bitmap.

setItem Replaces a specified menu item's style and representation.

```
virtual ISubmenu&
  setItem( const IMenuItem& menuItem );
```

Win	PM	Motif
Y	Y	Y

setText Replaces a specified menu item's text.

1
```
virtual ISubmenu&
  setText( unsigned long menuItemId,
           const char* newText );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ISubmenu&
  setText( unsigned long menuItemId,
           const IResourceId& newTextResId );
```

Win	PM	Motif
Y	Y	Y

Selection

Use selection members to determine if a menu item is selectable (enabled) or is displaying a selection state indicator. You can also change these properties. The ISubmenu class overrides these IMenu (p. 525) selection members so that information about how the menu was changed can be stored. The stored information allows the changes to be undone later.

checkItem Places a selection state indicator to the left of the item with the specified identifier.

```
virtual ISubmenu&
  checkItem( unsigned long itemId,
             Boolean checked = true );
```

Win	PM	Motif
Y	Y	Y

PM The selection state indicator is a check mark in front of the menu item.

Win The selection state indicator is a check mark in front of the menu item.

Motif The selection state indicator is a square-shaped object to the left of the specified item.

disableItem Makes the menu item with the specified identifier unselectable.

```
virtual ISubmenu&
  disableItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

enableItem Makes the menu item with the specified identifier selectable.

© IBM Corp. 1992, 1996

```
virtual ISubmenu&
  enableItem( unsigned long itemId,
              Boolean enabled = true );
```

Win	PM	Motif
Y	Y	Y

uncheckItem Removes the selection state indicator from the left of the item with the specified identifier.

```
virtual ISubmenu&
  uncheckItem( unsigned long itemId );
```

Win	PM	Motif
Y	Y	Y

PM The selection state indicator is a check mark in front of the menu item.

Win The selection state indicator is a check mark in front of the menu item.

Motif The selection state indicator is a square-shaped object to the left of the specified item.

Undo

Undo members support restoring the ISubmenu to the state that existed before the use of ISubmenu members. You usually do not have to call these functions because the IMenuHandler (p. 576) class calls them automatically.

undo Reverses all the changes made to the menu using the ISubmenu member functions.

```
virtual ISubmenu&
  undo();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IMenu		
add	enableNotification	resetDisabledBackgroundColor
addAsNext	foregroundColor	resetDisabledForegroundColor
addBitmap	hiliteBackgroundColor	resetForegroundColor
addItem	hiliteForegroundColor	resetHiliteBackgroundColor
addSeparator	id	resetHiliteForegroundColor
addSubmenu	isFrameWindow	selectItem
addText	isItemChecked	setBackgroundColor
backgroundColor	isItemEnabled	setBitmap
checkItem	isValid	setConditionalCascade
convertToGUIStyle	itemHelpId	**setDefaultStyle**

IMenu		
cursor	itemRect	setDisabledBackgroundColor
defaultStyle	menuHandle	setDisabledForegroundColor
deleteAt	menuItem	setForegroundColor
deleteItem	numberOfItems	setHiliteBackgroundColor
disabledBackgroundColor	owner	setHiliteForegroundColor
disabledForegroundColor	removeConditionalCascade	setItem
disableItem	removeSubmenu	setItemHelpId
elementAt	removeSubmenuAt	setSubmenu
enableItem	resetBackgroundColor	setText

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Inherited Public Data

IMenu		
classDefaultStyle	**noStyle**	

© IBM Corp. 1992, 1996

ISubmenu

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ISubmenu contains the following nested classes:

ISubmenu::Cursor (see page 931)

ISubmenu::Cursor

Derivation

IBase
  IVBase
    ISubmenu::Cursor

Inherited By None.

Header File isubmenu.hpp

Members

Member	Page	Member	Page
Constructor	931	setToLast	932
Cursor	931	setToNext	932
invalidate	932	setToPrevious	932
isValid	932	~Cursor	931
setToFirst	932		

The nested class ISubmenu::Cursor defines objects that you can use to iterate through the menu items in a submenu. In the same way that you can use a cursor to iterate through the objects in a collection, you can use this cursor object to iterate through a submenu.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor Constructs a cursor object for a submenu. You can only construct objects of this class from an object of the class ISubmenu (p. 919).

```
Cursor( const ISubmenu& menu );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The menu is invalid. Verify that the menu object exists.

~Cursor

© Copyright IBM Corp. 1992, 1996

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Cursor Movement

Use cursor movement members to set the cursor to a menu item in the submenu. The cursor must be set to an item in the submenu before it can be used to access, add, or remove menu items.

setToFirst Points the cursor to the first menu item.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToLast Points the cursor to the last menu item.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Points the cursor to the next menu item. If there is none, the cursor is invalidated.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Points the cursor to the previous menu item. If there is none, the cursor is invalidated.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation

Use cursor validation members to determine if the cursor is valid or to set the cursor to an invalid state. The cursor must be valid in order to use it to access an item.

invalidate Flags this cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Queries whether this cursor points to a valid menu item.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

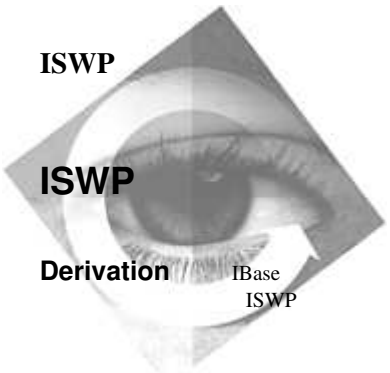
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



ISWP

Derivation IBase
ISWP

Inherited By None.

Header File iswp.hpp

Members

Member	Page	Member	Page
Constructor	934	setMove	936
behind	938	setNoAdjust	936
cx	938	setPosition	937
cy	938	setShow	937
fl	938	setSize	937
flags	936	setSizeFlag	937
hwnd	938	setZOrder	937
isHide	935	size	937
isMove	935	ulReserved1	939
isShow	935	ulReserved2	939
isSize	935	windowHandle	937
isZOrder	936	x	939
operator =	935	y	939
position	936	~ISWP	935
setHide	936		

The ISWP (set-window-position) class positions and sizes the client window and extension windows of the class IFrameWindow (p. 349). Classes derived from IFrameHandler (p. 342) may need to manipulate ISWP objects when overriding the IFrameHandler::format (p. 346) and IFrameHandler::positionExtensions (p. 348) functions. For these cases, ISWP objects can be accessed from the class IFrameFormatEvent (p. 339).

Public Functions

Constructors

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

ISWP

© Copyright IBM Corp. 1992, 1996

1 `ISWP();`

Win	PM	Motif
Y	Y	Y

The default constructor initializes all data to 0.

2 `ISWP( const ISWP& original );`

Win	PM	Motif
Y	Y	Y

Constructs a new object, as a copy of the specified ISWP object.

operator = Copies the specified ISWP object.

```
ISWP&
  operator =( const ISWP& original );
```

Win	PM	Motif
Y	Y	Y

~ISWP

```
~ISWP();
```

Win	PM	Motif
Y	Y	Y

Testing

Use testing members to determine the actions stored for a window.

isHide Returns if the window identified by the function windowHandle (p. 937) is to be hidden.

```
Boolean
  isHide() const;
```

Win	PM	Motif
Y	Y	I

isMove Returns if the window identified by the function windowHandle (p. 937) is to be moved. The function position (p. 936) returns the new position of the window.

```
Boolean
  isMove() const;
```

Win	PM	Motif
Y	Y	I

isShow Returns if the window identified by the function windowHandle (p. 937) is to be made visible.

```
Boolean
  isShow() const;
```

Win	PM	Motif
Y	Y	I

isSize Returns if the window identified by the function windowHandle (p. 937) is to be sized. The function size (p. 937) returns the new size for the window.

© IBM Corp. 1992, 1996

```
Boolean
  isSize() const;
```

Win	PM	Motif
Y	Y	I

isZOrder Returns if the order of the window identified by the function windowHandle (p. 937) is to be changed, relative to the order of its sibling windows.

```
Boolean
  isZOrder() const;
```

Win	PM	Motif
Y	Y	I

Window Manipulation

Use these members to identify actions to be applied to a window.

flags Returns the flags stored in this object.

```
unsigned long&
  flags();
```

Win	PM	Motif
Y	Y	Y

position Returns the point where the window identified by the function windowHandle (p. 937) is to be positioned, relative to its parent window.

```
IPoint
  position() const;
```

Win	PM	Motif
Y	Y	Y

setHide Specifies whether the window identified by the function windowHandle (p. 937) is to be hidden.

```
ISWP&
  setHide( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

setMove Specifies whether the window identified by the function windowHandle (p. 937) is to be moved.

```
ISWP&
  setMove( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

setNoAdjust Specifies whether the position or size of the window identified by the function windowHandle (p. 937) can be adjusted while it is moved or sized.

```
ISWP&
  setNoAdjust( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

setPosition Set the point where the window identified by the function windowHandle (p. 937) is to be positioned, relative to its parent window.

```
ISWP&
  setPosition( const IPoint& point );
```

Win	PM	Motif
Y	Y	Y

setShow Specifies whether the window identified by the function windowHandle (p. 937) is to be made visible.

```
ISWP&
  setShow( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

setSize Sets a new size for the window identified by the function windowHandle (p. 937).

```
ISWP&
  setSize( const ISize& size );
```

Win	PM	Motif
Y	Y	Y

setSizeFlag Specifies whether the window identified by the function windowHandle (p. 937) is to be sized.

```
ISWP&
  setSizeFlag( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

setZOrder Specifies whether the order of the window identified by the function windowHandle (p. 937) is to be changed, relative to its sibling windows.

```
ISWP&
  setZOrder( Boolean enable = true );
```

Win	PM	Motif
Y	Y	I

size Returns the new size for the window identified by the function windowHandle (p. 937).

```
ISize
  size() const;
```

Win	PM	Motif
Y	Y	Y

windowHandle

Returns the handle of the associated window.

```
IWindowHandle
  windowHandle() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

ISWP

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Public Data

Data

These data members store the state of the object.

behind Stores the window that the window identified by the function windowHandle (p. 937) is ordered after. Applying this action is equivalent to calling the function IWindow::positionBehindSibling (p. 1073).

```
IWindowHandle
  behind;
```

Win	PM	Motif
Y	Y	Y

cx Stores the width to be given to the window identified by the function windowHandle (p. 937). Applying this action is equivalent to calling the function IWindow::sizeTo (p. 1079).

```
long
  cx;
```

Win	PM	Motif
Y	Y	Y

cy Stores the height to be given to the window identified by the function windowHandle (p. 937). Applying this action is equivalent to calling the function IWindow::sizeTo (p. 1079).

```
long
  cy;
```

Win	PM	Motif
Y	Y	Y

fl Stores flags to indicate what actions are to be applied to the window identified by the function windowHandle (p. 937).

```
unsigned long
  fl;
```

Win	PM	Motif
Y	Y	Y

hwnd Stores the window that actions are to be applied to.

```
IWindowHandle
  hwnd;
```

Win	PM	Motif
Y	*Y*	*Y*

ulReserved1 Unused at this time.

```
unsigned long
  ulReserved1;
```

Win	PM	Motif
N	*Y*	*Y*

ulReserved2 Unused at this time.

```
unsigned long
  ulReserved2;
```

Win	PM	Motif
N	*Y*	*Y*

x Stores the x-component of where the window identified by the function windowHandle (p. 937) will be positioned on the screen relative to its parent window. Applying this action is equivalent to calling the function IWindow::moveTo (p. 1078).

```
long
  x;
```

Win	PM	Motif
Y	*Y*	*Y*

y Stores the y-component of where the window identified by the function windowHandle (p. 937) will be positioned on the screen relative to its parent window. Applying this action is equivalent to calling the function IWindow::moveTo (p. 1078).

```
long
  y;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ISWPArray

Derivation IBase
ISWPArray

Inherited By None.

Header File iswp.hpp

Members

Member	Page	Member	Page
Constructor	941	operator []	940
indexOf	940	size	941

The ISWPArray (set-window-position) class represents an array of ISWP (p. 934) objects. This class provides functions to do the following:

- Access a given ISWP object in the array
- Locate the ISWP object for a child window of the frame window, specified by window identifier

Public Functions

Array Contents

Use these members to operate on the collection of ISWP (p. 934) objects.

indexOf Returns the 0-based index of an ISWP (p. 934) element in the array. You can pass this index to ISWPArray::operator[] (p. 940) to access the ISWP object.

windowId The identifier of the window for the desired ISWP object.

```
unsigned
  indexOf( unsigned long windowId ) const;
```

Win	PM	Motif
Y	Y	Y

operator []

Returns a reference to the specified element of the array.

index The 0-based index of the ISWP (p. 934) array element.

© Copyright IBM Corp. 1992, 1996

```
ISWP&
  operator []( unsigned index );
```

Win	PM	Motif
Y	Y	Y

size Returns the dimension of the array, as specified on the constructor.

```
unsigned
  size() const;
```

Win	PM	Motif
Y	Y	Y

Constructors

You can construct and destruct objects of this class.

ISWPArray Constructs objects of this class from ISWP (p. 934) objects.

```
ISWPArray( ISWP* array,
           unsigned dimension );
```

Win	PM	Motif
Y	Y	Y

array A pointer to an ISWP object. This pointer is presumed to be the first element of the array. The function IFrameFormatEvent::swpArray (p. 340) returns such a value.

dimension The number of elements in the array.

PM You can create an ISWP* for *array* by casting a Presentation Manager PSWP pointer.

Inherited Public Functions

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

ISystemBitmapHandle

Derivation

IBase
IHandle
IBitmapHandle
ISystemBitmapHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	942
~ISystemBitmapHandle	942

The ISystemBitmapHandle class accesses system bitmap resources. A *system bitmap* is a special type of bitmap that is not loaded from a resource file.

Public Functions

Constructors

You can construct and destruct objects of this class.

ISystemBitmapHandle

Constructs objects of this class from an identifier for the specific system bitmap resource you want to associate with this handle. The enumeration Identifier (p. 943) provides the valid set of these resource identifiers.

```
ISystemBitmapHandle( Identifier bitmapId );
```

Win	PM	Motif
Y	*Y*	*I*

~ISystemBitmapHandle

```
~ISystemBitmapHandle();
```

Win	PM	Motif
Y	*Y*	*I*

© Copyright IBM Corp. 1992, 1996

Inherited Public Functions

IBitmapHandle		
operator =	operator Value	

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

Identifier

```
Identifier {
    systemMenu,                  systemMenuPressed,
    scrollBarUpArrow,            scrollBarUpArrowPressed,
    scrollBarUpArrowDisabled,    scrollBarDownArrow,
    scrollBarDownArrowPressed,   scrollBarDownArrowDisabled,
    scrollBarRightArrow,         scrollBarRightArrowPressed,
    scrollBarRightArrowDisabled, scrollBarLeftArrow,
    scrollBarLeftArrowPressed,   scrollBarLeftArrowDisabled,
    menuCheckMark,               menuAttached,
    checkBoxCheck,               comboBoxDownArrow,
    pushButtonCorners,           minimizeButton,
    minimizeButtonPressed,       maximizeButton,
    maximizeButtonPressed,       restoreButton,
    restoreButtonPressed,        childSystemMenu,
    childSystemMenuPressed,      drive,
    file,                        folder,
    collapsedTree,               expandedTree,
    program,                     sizeBox,
    smallSystemMenu
    };
```

© IBM Corp. 1992, 1996

ISystemBitmapHandle

Use these enumerators to specify system bitmap resources:

systemMenu
System menu

systemMenuPressed
System menu in pressed state

scrollBarUpArrow
Scroll bar up arrow

scrollBarUpArrowPressed
Scroll bar up arrow in pressed state

scrollBarUpArrowDisabled
Scroll bar up arrow in disabled state

scrollBarDownArrow
Scroll bar down arrow

scrollBarDownArrowPressed
Scroll bar down arrow in pressed state

scrollBarDownArrowDisabled
Scroll bar down arrow in disabled state

scrollBarRightArrow
Scroll bar right arrow

scrollBarRightArrowPressed
Scroll bar right arrow in pressed state

scrollBarRightArrowDisabled
Scroll bar right arrow in disabled state

scrollBarLeftArrow
Scroll bar left arrow

scrollBarLeftArrowPressed
Scroll bar left arrow in pressed state

scrollBarLeftArrowDisabled
Scroll bar left arrow in disabled state

menuCheckMark
Menu check mark

menuAttached
Cascading menu mark

checkBoxCheck
Check box or radio button check marks

comboBoxDownArrow
Combo box down arrow

pushButtonCorners
Push button corners

minimizeButton
Minimize button

minimizeButtonPressed
Minimize button in pressed state

maximizeButton
Maximize button

maximizeButtonPressed
Maximize button in pressed state

restoreButton
Restore button

restoreButtonPressed
.Restore button in pressed state

childSystemMenu
System menu for child windows

childSystemMenuPressed
System menu for child windows in pressed state

drive
Drive

file
File

folder
Folder

collapsedTree
Tree entry can be collapsed.

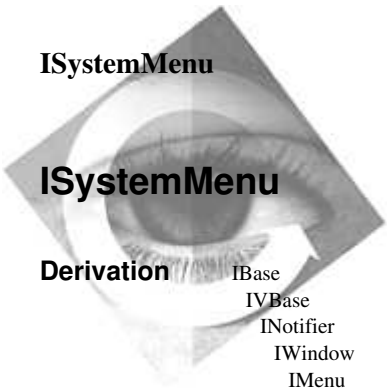
expandedTree
Tree entry can be expanded.

program
An executable file

sizeBox
Size box in bottom-right corner of a frame window

smallSystemMenu
Small system menu

© IBM Corp. 1992, 1996



ISystemMenu

Derivation

IBase
IVBase
INotifier
IWindow
IMenu
ISystemMenu

Inherited By None.

Header File isysmenu.hpp

Members

Member	Page	Member	Page
Constructor	947	idPulldown	950
idClose	949	idRestore	950
idHide	949	idSize	950
idMaximize	949	idWindowList	950
idMinimize	949	~ISystemMenu	947
idMove	949		

The ISystemMenu class enables you to manipulate your application's copy of the system menu. You can add, change, or remove items from the system menu for your application.

You can use the member functions in the classes IMenu (p. 525) and ISubmenu (p. 919) to change the ISystemMenu. You can also add objects of the class IMenuItem (p. 582) to the ISystemMenu.

The member functions IMenu::disableItem (p. 544) and ISubmenu::disableItem (p. 927) can have no effect in certain situations. For example, you cannot disable the **Size** menu item on the system menu if the frame window has a sizing border.

Win In the Windows operating system, ISystemMenu::idHide has the same effect as ISystemMenu::idMinimize.

AIX does not support the ISystemMenu::idClose (p. 949) member. In AIX, this is equivalent to the ClientMessage sent when the user selects Alt+F4 on a frame window.

In OS/2 Presentation Manager, this is the same as the SC_CLOSE system command. You can use ISystemMenu::idClose with ICommandHandler::systemCommand (p. 217). Within systemCommand, compare ISystemMenu::idClose to the event's

© Copyright IBM Corp. 1992, 1996

command ID. This is useful when you require a confirmation window for closing your application.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign ISystemMenu objects because both the copy constructor and assignment operator are private functions.

ISystemMenu

```
ISystemMenu( IFrameWindow* owner );
```

Win	PM	Motif
Y	*Y*	*I*

Constructs an ISystemMenu object for the system menu of the specified IFrameWindow (p. 349) object.

Exceptions	
IInvalidParameter	*owner* is 0. You must specify a valid IFrameWindow as the ISystemMenu owner.
IInvalidRequest	The specified frame window does not have a system menu.

~ISystemMenu

```
virtual
  ~ISystemMenu();
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Functions

IMenu		
add	enableNotification	resetDisabledBackgroundColor
addAsNext	foregroundColor	resetDisabledForegroundColor
addBitmap	hiliteBackgroundColor	resetForegroundColor
addItem	hiliteForegroundColor	resetHiliteBackgroundColor
addSeparator	id	resetHiliteForegroundColor
addSubmenu	isFrameWindow	selectItem
addText	isItemChecked	setBackgroundColor
backgroundColor	isItemEnabled	setBitmap

© IBM Corp. 1992, 1996

ISystemMenu

IMenu		
checkItem	isValid	setConditionalCascade
convertToGUIStyle	itemHelpId	**setDefaultStyle**
cursor	itemRect	setDisabledBackgroundColor
defaultStyle	menuHandle	setDisabledForegroundColor
deleteAt	menuItem	setForegroundColor
deleteItem	numberOfItems	setHiliteBackgroundColor
disabledBackgroundColor	owner	setHiliteForegroundColor
disabledForegroundColor	removeConditionalCascade	setItem
disableItem	removeSubmenu	setItemHelpId
elementAt	removeSubmenuAt	setSubmenu
enableItem	resetBackgroundColor	setText

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Item Identifiers

Use item identifier members when calling IMenu (p. 525) functions to affect items on the system menu.

When you want to add to the system menu pull-down, specify the ISystemMenu::idPulldown (p. 950) member for *intoSubmenuId* when calling IMenu::addItem (p. 529). For example:

```
sysmenu.addItem(myMenuItem, ISystemMenu::idPulldown);
```

If you do not specify ISystemMenu::idPulldown, *myMenuItem* appears on the title bar to the right of the system menu icon.

If you place additional menu items on the title bar, use bitmap items rather than text items.

Note: When adding a bitmap item to the title bar using IMenu::addBitmap (p. 527), *myBitmapHandle* needs to reference a bitmap whose dimensions are 40 x 36. For example:

```
sysmenu.addBitmap(myMenuItem, myBitmapHandle);
```

Additionally, if you add more that one bitmap to the right of the system menu icon, draw each bitmap so that its left edge provides a visual separator.

idClose Identifier of the **Close** system menu item.

```
static const unsigned long
  idClose;
```

Win	PM	Motif
Y	*Y*	*Y*

idHide Identifier of the **Hide** system menu item.

```
static const unsigned long
  idHide;
```

Win	PM	Motif
Y	*Y*	*I*

Win This identifier has the same value as the idMinimize (p. 949) identifier.

idMaximize Identifier of the **Maximize** system menu item.

```
static const unsigned long
  idMaximize;
```

Win	PM	Motif
Y	*Y*	*I*

idMinimize Identifier of the **Minimize** system menu item.

```
static const unsigned long
  idMinimize;
```

Win	PM	Motif
Y	*Y*	*I*

idMove Identifier of the **Move** system menu item.

```
static const unsigned long
  idMove;
```

Win	PM	Motif
Y	*Y*	*I*

© IBM Corp. 1992, 1996

ISystemMenu

idPulldown Identifier of the system menu pull-down. Use this identifier as the submenu identifier for the system menu.

```
static const unsigned long
  idPulldown;
```

Win	PM	Motif
Y	*Y*	*I*

idRestore Identifier of the **Restore** system menu item.

```
static const unsigned long
  idRestore;
```

Win	PM	Motif
Y	*Y*	*I*

idSize Identifier of the **Size** system menu item.

```
static const unsigned long
  idSize;
```

Win	PM	Motif
Y	*Y*	*I*

idWindowList

Identifier of the **Window list** system menu item.

```
static const unsigned long
  idWindowList;
```

Win	PM	Motif
Y	*Y*	*I*

Inherited Public Data

IMenu		
classDefaultStyle	**noStyle**	

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ISystemPointerHandle

Derivation

IBase
IHandle
IPointerHandle
ISystemPointerHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	952
~ISystemPointerHandle	952

The ISystemPointerHandle class accesses system pointer resources. A *system pointer* is a special type of pointer that is not loaded from a resource file.

Public Functions

Constructors

You can construct and destruct objects of this class.

ISystemPointerHandle

Constructs objects of this class from an identifier for the specific system-pointer resource you want to associate with this handle. The enumeration Identifier (p. 953) provides the valid set of these resource identifiers.

The constructor also accepts an optional Boolean, *makeCopy*, that specifies whether you want a copy of the system pointer to be made. If you want to modify the pointer, specify true. The default is false.

```
ISystemPointerHandle( Identifier pointerId,
                      Boolean makeCopy = false );
```

Win	PM	Motif
Y	*Y*	*I*

~ISystemPointerHandle

© Copyright IBM Corp. 1992, 1996

```
~ISystemPointerHandle();
```

Win	PM	Motif
Y	*Y*	*I*

Inherited Public Functions

IPointerHandle		
operator =	operator Value	

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

Identifier

```
Identifier {
    arrow,              text,               wait,
    move,               sizeNWSE,           sizeNESW,
    sizeHorizontal,     sizeVertical,       standardApplication,
    information,        question,           error,
    warning,            illegalOperation,   singleFile,
    multipleFile,       folder,             program,
    dragFade
    };
```

Use these enumerators to specify various system pointers:

arrow
Arrow pointer

text
Text I-beam pointer

© IBM Corp. 1992, 1996

ISystemPointerHandle

wait
Hourglass pointer

move
Move pointer

sizeNWSE
Downward-sloping, double-headed arrow pointer

sizeNESW
Upward-sloping, double-headed arrow pointer

sizeHorizontal
Horizontal, double-headed arrow pointer

sizeVertical
Vertical, double-headed arrow pointer

standardApplication
Standard application icon pointer

information
Information icon pointer

question
Question mark icon pointer

error
Exclamation mark icon pointer

warning
Warning icon pointer

illegalOperation
Illegal operation icon pointer

singleFile
Single file icon pointer

multipleFile
Multiple files icon pointer

folder
Folder icon pointer

program
Application program icon pointer

dragFade
Drag-fading-effect icon pointer

ITextControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl

Inherited By

IButton
ICircularSlider
IEntryField
IFlyText
IGroupBox
IMultiLineEdit
IStaticText
ITitle

Header File itextctl.hpp

Members

Member	Page	Member	Page
Constructor	958	text	957
clipboardHasTextFormat	955	textId	959
displaySize	957	textLength	957
setLayoutDistorted	956	~ITextControl	956
setText	956		

The ITextControl class is an abstract base class for all text control window classes. It provides functions that manipulate text strings.

Motif ITextControl is not the class corresponding to a text widget. If you are looking for a class for a text widget, see IEntryField (p. 271) and IMultiLineEdit (p. 641).

Public Functions

Clipboard Operations

Use these members to interface with the clipboard.

clipboardHasTextFormat

Returns true if the clipboard is in text format.

```
static Boolean
  clipboardHasTextFormat();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Motif If one of the registered formats for the clipboard data is of type "STRING," this function returns true.

Constructors

Because this class is an abstract base class, you cannot directly construct objects of this class. The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. You can destruct objects of this class.

~ITextControl

```
virtual
  ~ITextControl();
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout-support members are overrides that supply information used by the canvas classes to provide dialog-like behavior.

setLayoutDistorted

Processes a font change like a minimum size change. This is a virtual override of setLayoutDistorted (p. 1066).

```
virtual ITextControl&
  setLayoutDistorted( unsigned long layoutAttributeOn,
                      unsigned long layoutAttributeOff );
```

Win	PM	Motif
Y	Y	Y

Text Processing

Use these members to manage the text. You can query and set the text as well as query the length of the text for objects of this class.

setText

Sets the control window's text.

If the setText member function is used to set the text in an IComboBox (p. 182) constructed with the readOnlyDropDown style, non-null text specified must already exist in the combo box list. The item matching the new text will be selected. If the new text specified is the null string, all items in the list will be deselected.

1

```
virtual ITextControl&
  setText( const char* text );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextControl&
  setText( const IResourceId& text );
```

Win	PM	Motif
Y	Y	Y

text Returns the control window's text.

```
virtual IString
  text() const;
```

Win	PM	Motif
Y	Y	Y

textLength Returns the current length of the control window's text in bytes.

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

Win	PM	Motif
Y	Y	Y

Window Painting

Use these members to update the window.

displaySize Returns the width and height of the rectangle enclosing the string. If you do not specify a string, the size needed to contain the current text is returned.

text String to be enclosed.

```
virtual ISize
  displaySize( const char* text = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

PM This function uses the font attribute of the presentation space, including a sheared or rotated string. This function also supports double-byte character set (DBCS) characters.

Motif This member uses the underlying widget's current font list. It supports multiple-byte character sets but not sheared or rotated strings.

Exceptions	
IAccessError	The operating system's request to query the size of the text box failed.

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

© IBM Corp. 1992, 1996

ITextControl

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Protected Functions

Constructors

Because this class is an abstract base class, you cannot directly construct objects of this class. The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected. You can destruct objects of this class.

ITextControl

```
ITextControl();
```

Win	PM	Motif
Y	Y	Y

Derived classes use this protected constructor to create objects of this class.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

textId Notification identifier provided to observers when the text of a text control window changes.

```
static INotificationId const
  textId;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IControl		
group	**tabStop**	

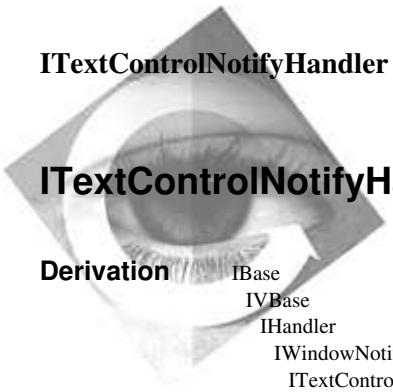
IWindow		
activeColorId	**disabledBackgroundColorId**	**noStyle**
backgroundColorId	**disabledForegroundColorId**	**positionId**
borderColorId	**enableId**	**saveBits**
clipChildren	**focusId**	**shadowColorId**
clipSiblings	**fontId**	**sizeId**
clipToParent	**foregroundColorId**	**synchPaint**
commandId	**hiliteBackgroundColorId**	**systemCommandId**
deleteId	**hiliteForegroundColorId**	**visible**
disabled	**inactiveColorId**	**visibleId**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



ITextControlNotifyHandler

Derivation

IBase
IVBase
IHandler
IWindowNotifyHandler
ITextControlNotifyHandler

Inherited By

IButtonNotifyHandler
ICircularSliderNotifyHandler
IEntryFieldNotifyHandler
IMultiLineEditNotifyHandler
ITitleNotifyHandler

Header File itextcnh.hpp

Members

Member	Page
Constructor	960
dispatchHandlerEvent	961
~ITextControlNotifyHandler	961

The ITextControlNotifyHandler class processes events for all classes of text controls.

This class is designed to handle events that require the text control class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ITextControlNotifyHandler

This is the default constructor and accepts no parameters.

```
ITextControlNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

~ITextControlNotifyHandler

```
virtual
  ~ITextControlNotifyHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

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 the text control observers if the following event is received:

text event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

© IBM Corp. 1992, 1996

ITextControlNotifyHandler

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ITextSpinButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
IBaseSpinButton
ITextSpinButton

Inherited By None.

Header File ispintxt.hpp

Members

Member	Page	Member	Page
Constructor	964	removeAll	969
add	968	removeAt	970
addAsFirst	968	replaceAt	970
addAsLast	969	setDefaultStyle	968
addAsNext	969	setLimit	966
classDefaultStyle	972	setText	967
convertToGUIStyle	967	spinDown	966
defaultStyle	967	spinTo	970
elementAt	969	spinUp	966
enableNotification	966	text	967
isSpinFieldValid	966	~ITextSpinButton	965

The ITextSpinButton class creates and manages text spin button controls. The text spin button maintains an array of text data.

Handlers derived from the following classes can handle events for ITextSpinButton objects:

IEditHandler (p. 267)
IFocusHandler (p. 322)
IKeyboardHandler (p. 490)
IMouseHandler (p. 631)
IPaintHandler (p. 706)
IResizeHandler (p. 802)
ISpinHandler (p. 896)

Motif The ITextSpinButton constructor creates objects of this class using the following Motif widgets:

- An XmForm widget is created with a single line XmText child.

© Copyright IBM Corp. 1992, 1996

- If the spin button has the style IBaseSpinButton::master, an XmForm widget containing two XmArrowButton widgets is also created as a child of the top XmForm. IWindow::handle returns the handle of the XmText widget.

The User Interface Class Library provides the behavior of an ITextSpinButton object via private callbacks and a default handler. The ITextSpinButton class uses a default handler derived from the class IKeyboardHandler. Therefore, attach user-defined handlers derived from IKeyboardHandler to the ITextSpinButton object rather than to its owner window. This enables events to be dispatched to user-defined handlers before the default handler.

Handlers derived from IEditVerifyHandler can be attached to ITextSpinButton objects.

The AIX environment supports only the first constructor, which creates an object of this class from a control ID, parent and owner windows, a rectangle, and a style.

Public Functions

Constructors

You can construct and destruct objects of this class.

ITextSpinButton

1 `ITextSpinButton( unsigned long id,`
`IWindow* parent,`
`IWindow* owner,`
`const IRectangle& initial = IRectangle ( ),`
`const Style& style = defaultStyle ( ) );`

Win	PM	Motif
Y	*Y*	*Y*

You can construct objects of this class by creating the specified text spin-button control and an object for it.

windowId A unique ID for the text spin button control.

parent The parent window.

owner The owner window.

initial A rectangle defining the size and placement of the text spin button window. Optional.

style The style of the control. Optional.

Exceptions	
IInvalidRequest.	The style is for a numeric-only spin button, which is invalid for an ITextSpinButton.

2 `ITextSpinButton( unsigned long id, IWindow* parent );`

Win	PM	Motif
Y	Y	N

You can construct objects of this class by creating the object for the specified text spin-button control.

id A unique ID for the text spin button control.
parent The parent dialog window.

Win This constructor can only be used to create an object with the pmCompatible (p. 118) style.

Exceptions	
IInvalidRequest.	The ID is for a numeric only spin button, which is invalid for an ITextSpinButton.

3 `ITextSpinButton( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	N

You can construct objects of this class by creating the object for the specified text spin-button control.

handle The window handle of an existing text spin-button control.

Win This constructor can only be used to create an object with the pmCompatible (p. 118) style.

Exceptions	
IInvalidRequest.	The handle is for a numeric-only spin button, which is invalid for an ITextSpinButton.

~ITextSpinButton

```
virtual
  ~ITextSpinButton();
```

Win	PM	Motif
Y	Y	Y

Limit and Spin

Use these members to manage the spin field for objects of this class.

© IBM Corp. 1992, 1996

setLimit Sets the number of characters permitted in the spin field. The User Interface Class Library defines this limit as 255 at the time of construction.

Note: If you call this function with a limit that does not display all the numbers in the spin button range, erratic results can occur when the button is spun. For example, if the range is 1 to 100 and the limit is set to 2, the button spins up to 99 and wraps to 10 instead of 1. Spinning down, the button wraps from 1 to 10.

```
virtual ITextSpinButton&
  setLimit( unsigned long aNumber );
```

Win	PM	Motif
N	N	Y

spinDown Spins the button down the specified number of times.

```
virtual ITextSpinButton&
  spinDown( unsigned long spinBy = 1 );
```

Win	PM	Motif
N	N	Y

spinUp Spins the button up the specified number of times.

```
virtual ITextSpinButton&
  spinUp( unsigned long spinBy = 1 );
```

Win	PM	Motif
N	N	Y

Notification Members

Use these members to enable notifications sent to observer objects.

enableNotification

Causes the text spin-button control to send notifications to any observer objects added.

```
virtual ITextSpinButton&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Set Text and Validate

Use these members to query and modify the text of the spin button and to validate the contents of the spin field.

isSpinFieldValid

If the contents of the spin field matches one of the text values in the text array, true is returned. If *caseSensitive* is set to true, the value in the text array must match exactly.

```
virtual Boolean
  isSpinFieldValid( Boolean caseSensitive = false ) const;
```

Win	PM	Motif
Y	Y	Y

setText Sets the displayed contents of the spin field, regardless of the validity of the text. This does not alter the contents of the spin button array.

1
```
virtual ITextSpinButton&
  setText( const char* string );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  setText( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

text Returns the displayed contents of the spin field.

```
virtual IString
  text() const;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the ITextSpinButton (p. 963) class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IBaseSpinButton Styles (p. 116)

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	N

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 972) unless you have changed the style using setDefaultStyle (p. 968).

```
static Style
  defaultStyle();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

setDefaultStyle

Sets the default style for all subsequent text spin buttons.

style Use the styles provided by ITextSpinButton Styles (p. 972) to specify the default style.

```
static void
  setDefaultStyle( const Style& style );
```

Win	PM	Motif
Y	Y	Y

Text List Operations

Use these members to manage the spin button object's text array.

add

Adds a new item at the cursor or index position. One of the overloaded versions of this function lets you add an array of text strings to the spin button.

1
```
virtual ITextSpinButton&
  add( const char* string,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  add( const IResourceId& item,
       Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The cursor is invalid.

3
```
virtual ITextSpinButton&
  add( const char* string,
       unsigned long index );
```

Win	PM	Motif
Y	Y	Y

4
```
virtual ITextSpinButton&
  add( const IResourceId& item,
       unsigned long index );
```

Win	PM	Motif
Y	Y	Y

5
```
virtual ITextSpinButton&
  add( const char * const* stringArray,
       unsigned long index,
       unsigned long count );
```

Win	PM	Motif
Y	Y	Y

addAsFirst

Adds the item as the first item. You can use this function to add an item to an empty ITextSpinButton object.

1
```
virtual ITextSpinButton&
  addAsFirst( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  addAsFirst( const char* string );
```

Win	PM	Motif
Y	Y	Y

addAsLast Adds the item as the last item. You can use this function to add an item to an empty text ITextSpinButton object.

1
```
virtual ITextSpinButton&
  addAsLast( const IResourceId& item );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  addAsLast( const char* string );
```

Win	PM	Motif
Y	Y	Y

addAsNext Adds the item following the current cursor position and sets the cursor on it.

1
```
virtual ITextSpinButton&
  addAsNext( const char* string,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  addAsNext( const IResourceId& item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

elementAt Returns the string at the cursor or index position.

1
```
virtual IString
  elementAt( const Cursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The cursor is invalid.

2
```
virtual IString
  elementAt( unsigned long index ) const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The index is invalid.

removeAll Removes all items in the spin button.

© IBM Corp. 1992, 1996

```
virtual ITextSpinButton&
  removeAll();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system's request to reset the spin button has failed.

removeAt Removes the item at the cursor position and sets the cursor to the item following the removed item. If the last item is removed, the cursor is invalidated.

```
virtual ITextSpinButton&
  removeAt( Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The cursor is invalid.

replaceAt Replaces the item at the cursor position.

1
```
virtual ITextSpinButton&
  replaceAt( const IResourceId& item,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

2
```
virtual ITextSpinButton&
  replaceAt( const char* newString,
             Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The cursor is invalid.

spinTo Spins the button to the specified cursor or index position.

This function is overloaded with a version that takes a text value (*string*) and, optionally, a Boolean flag for controlling case sensitivity. If the *string* specified is valid, this version spins the button to *string*. If *caseSensitive* is set to true, the *string* must match exactly one of the values of the spin-button text array.

1
```
virtual ITextSpinButton&
  spinTo( const Cursor& cursor );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The cursor is invalid.

2
```
virtual ITextSpinButton&
  spinTo( unsigned long index );
```

Win	PM	Motif
Y	Y	Y

3
```
virtual ITextSpinButton&
  spinTo( const char* string,
          Boolean caseSensitive = false );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The specified string cannot be found in the spin button.

Inherited Public Functions

IBaseSpinButton		
addBorder	hasBorder	resetBackgroundColor
alignment	isFastSpinEnabled	resetForegroundColor
backgroundColor	isMaster	setAlignment
convertToGUIStyle	isPMCompatible	setBackgroundColor
disableDataUpdate	isServant	setForegroundColor
disableFastSpin	isSpinFieldValid	setLimit
enable	isWriteable	setMaster
enableDataUpdate	limit	spinDown
enableFastSpin	moveSizeTo	spinUp
foregroundColor	removeBorder	topHandle

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

ITextSpinButton

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Note: Members of IWindow (p. 1044) are not shown.

Inherited Protected Functions

IBaseSpinButton		
calcMinimumSize	initialize	registerCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Styles

These style members provide a set of valid styles for the ITextSpinButton (p. 963) class. You can use these styles with the styles in the following classes:

IWindow Styles (p. 1093)
IControl Styles (p. 224)
IBaseSpinButton Styles (p. 116)

classDefaultStyle

Provides the original default style for this class, which is the following: IBaseSpinButton::master | IBaseSpinButton::leftAlign | IBaseSpinButton::border3D | IWindow::visible.

```
static const Style
  classDefaultStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Public Data

IBaseSpinButton		
border3D	**master**	**rightAlign**

IBaseSpinButton		
centerAlign	noBorder	servant
fastSpin	pmCompatible	textId
leftAlign	readOnly	valueId

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 (p. 1044) are not shown.

Inherited Protected Data

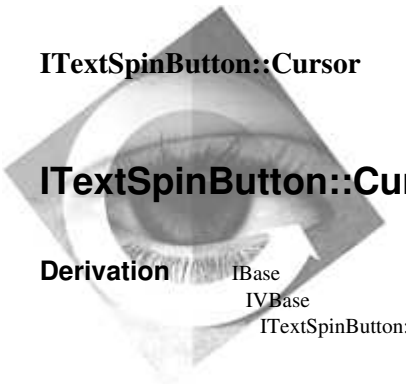
IBase		
recoverable	unrecoverable	

Nested Classes

ITextSpinButton contains the following nested classes:

ITextSpinButton::Style (see page 977)
ITextSpinButton::Cursor (see page 974)

© IBM Corp. 1992, 1996



ITextSpinButton::Cursor

Derivation

IBase
  IVBase
    ITextSpinButton::Cursor

Inherited By None.

Header File ispintxt.hpp

Members

Member	Page	Member	Page
Constructor	974	setToLast	975
Cursor	974	setToNext	975
invalidate	975	setToPrevious	975
isValid	975	~Cursor	974
setToFirst	975		

The ITextSpinButton::Cursor class creates and manages the cursor for an ITextSpinButton (p. 963) object. In the same way that you can use a cursor to traverse the objects in a collection, you can use this cursor to traverse the text items in a text spin button, one item at a time.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor You can construct objects of this class from an object of ITextSpinButton (p. 963).

```
Cursor( const ITextSpinButton& spinButton );
```

Win	PM	Motif
Y	Y	Y

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Cursor Movement

Use these members to change the cursor position.

setToFirst Points to the first item in the text list.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToLast Points to the last item in the text list.

```
virtual Boolean
  setToLast();
```

Win	PM	Motif
Y	Y	Y

setToNext Points to the next item in the text list. If there is none, the cursor is invalidated.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

setToPrevious

Points to the previous item in the text list. If there is none, the cursor is invalidated.

```
virtual Boolean
  setToPrevious();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation

Use these members to check and reset the validity of the cursor.

invalidate Flags this cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Queries whether this cursor points to a valid item.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

© IBM Corp. 1992, 1996

ITextSpinButton::Cursor

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ITextSpinButton::Style

Derivation

IBase
  IBitFlag
    ITextSpinButton::Style

Inherited By None.

Header File ispintxt.hpp

The nested class ITextSpinButton::Style provides a set of valid styles for the ITextSpinButton (p. 963) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

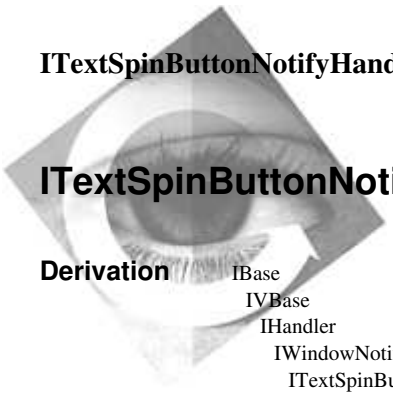
Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© Copyright IBM Corp. 1992, 1996



ITextSpinButtonNotifyHandler

Derivation

IBase
IVBase
IHandler
IWindowNotifyHandler
ITextSpinButtonNotifyHandler

Inherited By None.

Header File ispbttnh.hpp

Members

Member	Page
Constructor	978
dispatchHandlerEvent	979
~ITextSpinButtonNotifyHandler	978

The ITextSpinButtonNotifyHandler class processes events for all classes of text spin buttons.

This class is designed to handle events that require the text spin button class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ITextSpinButtonNotifyHandler

This is the default constructor and accepts no parameters.

```
ITextSpinButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~ITextSpinButtonNotifyHandler

© Copyright IBM Corp. 1992, 1996

```
virtual
  ~ITextSpinButtonNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

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 the text spin button observers if the following event is received:

- spin button change event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

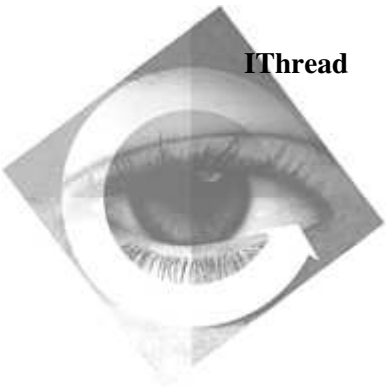
© IBM Corp. 1992, 1996

ITextSpinButtonNotifyHandler

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	



IThread

Derivation

IBase
  IVBase
    IThread

Inherited By

ICurrentThread

Header File

ithread.hpp

Members

Member	Page	Member	Page
Constructor	982	relatedHandlesList	988
adjustPriority	997	resume	992
asDebugInfo	986	setAutoInitGUI	987
asString	986	setDefaultAutoInitGUI	987
autoInitGUI	986	setDefaultQueueSize	990
current	995	setDefaultStackSize	991
currentHandle	995	setPriority	998
currentId	995	setQueueSize	990
defaultAutoInitGUI	986	setRelatedHandlesList	988
defaultQueueSize	988	setStackSize	991
defaultStackSize	990	setVariable	996
dialogControls	988	setWindowList	988
handle	996	stackSize	992
id	996	start	992
isStarted	996	startedThread	999
messageQueue	989	stop	994
newStartedThread	999	stopProcessingMsgs	987
operator =	998	suspend	995
priorityClass	997	variable	996
priorityLevel	997	windowList	988
queueSize	989	~IThread	985

The IThread class represents threads of execution within the current program. You can use this class to implement multithreaded applications. Objects of this class give you access to all of the tasking APIs of the operating system. In addition, these objects serve as the anchor for thread-specific information.

Generally, you use objects of this class in one of the following ways:

- To apply thread functions to the current thread. In most cases, these functions are applied to the IThread object reference returned by the static member function IThread::current (p. 995).

© Copyright IBM Corp. 1992, 1996

- To create additional threads of execution by creating new objects of this class and starting them.
- To manipulate threads of execution initiated using alternate means, for which only the thread's identifier is known.

PM Compile multithreaded programs (those that call the IThread::start (p. 992) member function) with /Gm+ to avoid unresolved externals at link time.

W32s Win32s does not support multiple threads per process. Applications running on this environment must be limited to a single thread.

Creating instances of this class or calling its member functions will not create and start threads on the system. Attempting to start a thread will result in an exception.

Motif The AIX release of the User Interface Class Library does not support multithreading. The only access that an AIX application should need to IThread is exported through the class ICurrentApplication (p. 237) with the call to ICurrentApplication::run (p. 240). This call starts the processing of messages for the application.

You cannot control the priority of the AIX thread. The main thread of execution is a process. The only control that a user has over the priority of a process is using the nice values, which are not supported.

The AIX release of the User Interface Class Library does not presently support multiple threads per process. Portable applications must be limited to a single thread.

The OS/2 version of IThread has functions to control the size of the message queue and the stack size associated with a thread. These functions are not needed in the AIX environment and have no effect on the execution of a thread. You can call these functions in the AIX environment with no adverse side effects. Therefore, they have been left in the interface for ease of porting applications to multiple platforms.

Public Functions

Constructors

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

IThread

1 `IThread();`

Win	PM	Motif
Y	*Y*	*Y*

Use this, the default constructor, to create an object of this class so that you can subsequently start using IThread::start (p. 992).

W32s Creating an instance of IThread with this constructor does not create or start a thread in the system.

2

```
IThread( const IThread& thread );
```

Win	PM	Motif
Y	Y	Y

thread Reference to an object of this class.

Use this constructor if you would like to make a copy of an existing object of this class. This is commonly referred to as a copy constructor.

W32s Creating an instance of IThread with this constructor does not create or start a thread in the system.

3

```
IThread(
   const IThreadId& threadID,
   const IThreadHandle& threadHandle =
     IThreadHandle::noHandle );
```

Win	PM	Motif
Y	Y	Y

threadID Reference to a thread identifier of a previously started thread.
threadHandle
Optional reference to a thread handle of a previously started thread.

Use this constructor to create an object of this class from the thread identifier (ID) of a previously started thread. You can use this constructor to provide this class' functionality to threads created using alternate means (for example, the native operating system's system calls).

PM The *threadHandle* parameter is ignored.

Win You must specify the *threadHandle* argument if the *threadId* argument does not refer to the current thread. You can use the return value of the CreateThread API for the thread handle.

W32s Creating an instance of IThread with this constructor does not create or start a thread in the system.

4

```
IThread( const IReference < IThreadFn >& threadFunction,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	Y	N

threadFunction
Reference to an object of the IReference template class that was created using a derived IThreadFn class.

© IBM Corp. 1992, 1996

autoInitGUI
Boolean value that you use to specify if the thread is a GUI thread.

Use this constructor to create an object of this class when you need to specify a function to run that you have defined by deriving a new class from the IThreadFn (p. 1004) class. This constructor passes the derived IThreadFn using an object of the IReference (Vol. I) class so that the derived IThreadFn can be deleted (if necessary) when the thread terminates.

Use this form of the constructor to create a new object of this class and immediately dispatch it. This is equivalent to using the default constructor and then dispatching the thread using IThread::start (p. 992).

This constructor permits you to specify whether or not the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

W32s Creating an instance of IThread with this constructor throws an exception.

Exceptions	
IInvalidRequest	The thread cannot be started; Win32s does not support multiple threads.

5
```
IThread( OptlinkFnPtr function,
         void* functionArgument,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	Y	N

function Pointer to a function that uses the _Optlink calling convention.

functionArgument
Pointer to a parameter that you are passing to the function pointed to by *function*.

autoInitGUI
Boolean value that you use to specify that the thread is a GUI thread.

Use this constructor to create an object of this class when you need to specify a function to run that is defined with the _Optlink calling convention. Such functions are typically started using the function _beginThread.

Use this form of the constructor to create a new object of this class and immediately dispatch it. This is equivalent to using the default constructor and then dispatching the thread using IThread::start (p. 992).

This constructor permits you to specify whether the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

W32s Creating an instance of IThread with this constructor throws an exception.

Exceptions	
IInvalidRequest	The thread cannot be started; Win32s does not support multiple threads.

6

```
IThread( SystemFnPtr function,
         unsigned long functionArgument,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	*Y*	*N*

function Pointer to a function that uses the _System calling convention.

functionArgument
Unsigned long parameter that you are passing to the function pointed to by *function*.

autoInitGUI
Boolean value that you use to specify that the thread is a GUI thread.

Use this constructor to create an object of this class when you need to specify a function to run that you have defined using the _System calling convention. Such functions are typically started using the thread APIs that are defined for the operating system.

Use this form of the constructor to create a new object of this class and immediately dispatch it. This is equivalent to using the default constructor and then dispatching the thread using IThread::start (p. 992).

This constructor permits you to specify whether or not the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

W32s Creating an instance of IThread with this constructor throws an exception.

Exceptions	
IInvalidRequest	The thread cannot be started; Win32s does not support multiple threads.

~IThread

Deallocates thread-related resources.

Note: When objects of this class are destructed, the thread is not terminated.

© IBM Corp. 1992, 1996

```
virtual
  ~IThread();
```

Win	PM	Motif
Y	Y	Y

W32s This member function has no effect.

Diagnostics

Use these members for diagnostic purposes. They return an IString representation of an object of this class.

asDebugInfo Provides debugging information about the class object. Use it to return general diagnostic information about the thread.

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString Provides debugging information about the class object. Use it to return a string of the form "IThread(tid)", where *tid* represents the thread identifier.

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Graphical User Interface (GUI) Support

Use these members to query and set the basic GUI support for threads.

autoInitGUI Determines if this thread's GUI support is automatically initialized.

```
virtual Boolean
  autoInitGUI() const;
```

Win	PM	Motif
Y	Y	Y

PM This member obtains the setting of the flag that automatically initializes Presentation Manager for this thread.

defaultAutoInitGUI

Determines the default GUI support for automatic initialization. Unless the support is explicitly set using IThread::setDefaultAutoInitGUI (p. 987), this function returns true for threads running in a GUI session and false for threads running in a non-GUI session.

```
static Boolean
  defaultAutoInitGUI();
```

Win	PM	Motif
Y	Y	Y

Win Unless the support is explicitly set using IThread::setDefaultAutoInitGUI (p. 987), this function always returns true.

setAutoInitGUI

Sets the automatic initialization state for this thread. Set *initFlag* to true, the default, for GUI-based threads and to false for non-GUI-based threads.

```
virtual IThread&
  setAutoInitGUI( Boolean initFlag = true );
```

Win	PM	Motif
Y	Y	Y

initFlag Boolean value that determines if GUI support is automatically initialized for this thread.

Motif Regardless of the value set by this function, the GUI is always initialized in AIX because of the User Interface Class Library's single-threaded limitation.

setDefaultAutoInitGUI

Sets the default GUI support for automatic initialization. Set *initFlag* to true, the default, for GUI-based threads and to false for non-GUI-based threads.

```
static void
  setDefaultAutoInitGUI( Boolean initFlag = true );
```

Win	PM	Motif
Y	Y	Y

initFlag Boolean value that determines if GUI support is automatically initialized for this thread.

Motif Regardless of the value set by this function, the GUI is always initialized in AIX because of the User Interface Class Library's single-threaded limitation.

stopProcessingMsgs

Terminates the processing of window events (messages) initiated by ICurrentThread::processMsgs (p. 248).

```
virtual IThread&
  stopProcessingMsgs();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The event processing was not terminated. The termination failed because the event processing was not initiated by ICurrentThread::processMsgs.

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library and are not intended for use by users of this class.

© IBM Corp. 1992, 1996

IThread

dialogControls

Obtains a pointer to the dialog controls list for this thread.

```
IDialogControls*
  dialogControls() const;
```

Win	PM	Motif
Y	N	N

relatedHandlesList

Obtains a pointer to the related handles list for this thread.

```
IRelatedHandlesList*
  relatedHandlesList() const;
```

Win	PM	Motif
N	N	Y

setRelatedHandlesList

Sets a pointer to the related handles list for this thread.

```
IThread&
  setRelatedHandlesList( IRelatedHandlesList* list );
```

Win	PM	Motif
N	N	Y

list Pointer to an IRelatedHandlesList object to set for this thread.

setWindowList

Sets a pointer to the window information list for this thread.

```
IThread&
  setWindowList( IWindowList* list );
```

Win	PM	Motif
Y	Y	Y

list Pointer to an IWindowList object to set for this thread.

windowList

Obtains a pointer to the window information list for this thread.

```
IWindowList*
  windowList() const;
```

Win	PM	Motif
Y	Y	Y

Message Queue

Use these members to query and set message queue information for graphical user interface (GUI) support.

defaultQueueSize

Obtains the default message queue size for the threads. The User Interface Class Library default queue size is 30.

Note: The User Interface Class Library uses this value by default for new objects of this class.

```
static long
  defaultQueueSize();
```

Win	PM	Motif
I	*Y*	*I*

PM This member obtains the default Presentation Manager message queue size to use for new IThreads.

Win This function returns the queue size that is set by IThread::setDefaultQueueSize (p. 990).

Motif You cannot control the queue size on the X environment. This function returns the queue size that is set using IThread::setDefaultQueueSize (p. 990). Initially, this value is 0.

messageQueue

Returns the handle of the message queue. A 0 is returned if the handle does not exist or is invalid.

1
```
IMessageQueueHandle
  messageQueue();
```

Win	PM	Motif
Y	*Y*	*I*

Use this version of the function if you want to reset an invalid message queue handle to 0. Do this if the handle is determined to be invalid.

2
```
IMessageQueueHandle
  messageQueue() const;
```

Win	PM	Motif
Y	*Y*	*I*

Use this version of the function if you want the invalid message queue handle to remain invalid.

queueSize

Obtains the message queue size for the thread. This is the thread used by the application to receive events from the operating system.

```
virtual long
  queueSize() const;
```

Win	PM	Motif
I	*Y*	*I*

Win This function returns the queue size that is set by IThread::setQueueSize (p. 990).

Motif You cannot control the size of the message queue in the X environment. This function returns the value set by IThread::setQueueSize (p. 990). The default value is returned by IThread::defaultQueueSize (p. 988).

© IBM Corp. 1992, 1996

setDefaultQueueSize

Sets the default message queue size for threads. The User Interface Class Library default queue size is 30.

Note: The User Interface Class Library uses this value by default for new objects of this class.

```
static void
  setDefaultQueueSize( long queueSize );
```

Win	PM	Motif
I	*Y*	*I*

aSize Long value that represents the default message queue size.

Win This function has no effect on the message queue size. The value set by this function is the value returned by IThread::defaultQueueSize (p. 988).

Motif You cannot control the size of the queue in the X environment. This function has no effect on the message queue size. The value set by this function is the value returned by IThread::defaultQueueSize (p. 988).

setQueueSize

Sets the GUI message queue size for this thread.

```
virtual IThread&
  setQueueSize( long queueSize );
```

Win	PM	Motif
I	*Y*	*I*

queueSize Long value that represents the new message queue size.

Win This function has no effect on the size of the message queue. The value set by this function is the value returned by IThread::queueSize (p. 989).

Motif You cannot control the message queue size for X. This function has no effect on the size of the message queue. The value set by this function is the value returned by IThread::queueSize (p. 989).

Stack Size

Use these members to query and set this thread's stack size.

defaultStackSize

Obtains the default stack size for threads in bytes.

Note: The User Interface Class Library uses this value by default for new objects of this class.

```
static unsigned long
  defaultStackSize();
```

Win	PM	Motif
Y	*Y*	*I*

W32s You cannot control the stack size for a thread in this environment. This function returns the stack size that is set using IThread::setDefaultStackSize (p. 991). Initially, this value is 32768.

Motif You cannot control the stack size for a thread in the X environment. This function returns the stack size that is set using IThread::setDefaultStackSize (p. 991). Initially, this value is 32768.

setDefaultStackSize

Sets the default stack size for threads in bytes.

Note: The User Interface Class Library uses this value by default for new objects of this class.

```
static void
  setDefaultStackSize( unsigned long size );
```

Win	PM	Motif
Y	*Y*	*I*

size Unsigned long value that represents the default stack size in bytes.

W32s You cannot control the size of the stack in this environment. This function has no effect on the size of the stack. The value set by this function is the value returned by IThread::defaultStackSize (p. 990).

Motif You cannot control the size of the stack in the AIX environment. This function has no effect on the size of the stack. The value set by this function is the value returned by IThread::defaultStackSize (p. 990).

setStackSize

Sets this thread's stack size in bytes. If you have already started the thread, this value takes effect only when the thread is stopped and restarted.

```
virtual IThread&
  setStackSize( unsigned long size );
```

Win	PM	Motif
Y	*Y*	*I*

size Unsigned long value that represents the stack size in bytes.

Win This value controls the amount of storage which will be committed to the thread by the system when the thread is started. Additional storage, up to the maximum available to your program, will be automatically allocated by the system if needed. You can control the maximum available stack available to your program by using the /STACKSIZE linker directive.

W32s You cannot control the stack size in this environment. This function has no effect on the size of the stack used. The value set by this function is the value returned by IThread::stackSize (p. 992).

© IBM Corp. 1992, 1996

Motif You cannot control the stack size in AIX. This function has no effect on the size of the stack used. The value set by this function is the value returned by IThread::stackSize (p. 992).

stackSize Obtains this thread's stack size in bytes.

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

Win	PM	Motif
Y	*Y*	*I*

Win Returns the value of the stack size that was set with the function IThread::setStackSize (p. 991). This value represents the stack size used as the initial stack size when the thread is created. For the primary thread, its value is the same as the initial default stack size. This value does not represent the actual committed stack in use by the thread.

You can control the maximum available stack available to your program by using the /STACKSIZE linker directive.

W32s You cannot control the stack size in this environment. This is not the actual size of the stack. This function returns the value of the stack size that was set with the function IThread::setStackSize (p. 991).

Motif Returns the value of the stack size that was set with the function IThread::setStackSize (p. 991). You cannot control the size of the stack in AIX; therefore, this is not actually the size of the stack. The default value is returned by IThread::defaultStackSize (p. 990).

Starting and Stopping Threads

Use these members to start or stop threads.

resume Resumes the thread's execution.

```
virtual void
  resume();
```

Win	PM	Motif
Y	*Y*	*I*

W32s This member function has no effect.

Motif The AIX release of the User Interface Class Library supports only a single-threaded environment. Therefore, you cannot suspend or resume a thread.

Exceptions	
IAccessError	The thread was not resumed. The thread was not suspended or the thread identifier (ID) is invalid.

start Starts an asynchronous thread.

1

```
void
  start( const IReference < IThreadFn >& threadFunction,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	*Y*	*N*

threadFunction
Reference to an object of the IReference template class that was created using a derived IThreadFn class.

autoInitGUI
Boolean value that you use to specify if the thread is a GUI thread.

You can start threads with a derived IThreadFn (p. 1004) class. This function passes the derived IThreadFn using an object of the IReference (Vol. I) class so that the derived IThreadFn can be deleted (if necessary) when the thread terminates.

This function permits you to specify whether the thread is started as a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

PM The OS/2 version also allows threads to be started via the following means:

- With an *IThread::OptlinkFnPtr* type and *void** parameter. This is to provide support for functions otherwise started using _beginthread.
- With an *IThread::SystemFnPtr* type and *unsigned long* parameter. This is to provide support for functions otherwise started using DosCreateThread.

W32s This member function has no effect.

Motif Because the User Interface Class Library supports only single threading, there is no need to call this function.

2

```
void
  start( OptlinkFnPtr function,
         void* functionArgument,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	*Y*	*N*

function Pointer to a function that uses the _Optlink calling convention.

functionArgument
Pointer to a parameter that you are passing to the function pointed to by *pfn*.

autoInitGUI
Boolean value that you use to specify if the thread is a GUI thread.

You can start threads with a function that is defined using the _Optlink calling

© IBM Corp. 1992, 1996

convention. Such functions are typically started using the function, _beginThread. This function gives you the same capability.

This constructor permits you to specify whether the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

W32s This member function has no effect.

3

```
void
  start( SystemFnPtr function,
         unsigned long functionArgument,
         Boolean autoInitGUI = defaultAutoInitGUI ( ) );
```

Win	PM	Motif
Y	Y	N

function Pointer to a function that uses the _System calling convention.

functionArgument
Unsigned long parameter that you are passing to the function pointed to by *pfn*.

autoInitGUI
Boolean value that you use to specify if the thread is a GUI thread.

You can start threads with a function that is defined using the _System calling convention. Such functions are typically started using the thread APIs that are defined for the operating system. This function gives you the same capability.

This constructor permits you to specify whether the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (p. 247) is called automatically after the thread is started.

W32s This member function has no effect.

stop

Stops the thread.

```
virtual void
  stop();
```

Win	PM	Motif
Y	Y	Y

PM Do not use this function on suspended threads or the results can be unpredictable. This function is very powerful. Use it with extreme caution, preferably when the state of the target thread is known.

W32s This member function has no effect.

Exceptions	
IAccessError	The thread was not stopped. The thread was either busy or the thread identifier (ID) is invalid.

suspend Suspends the thread's execution.

```
virtual void
  suspend();
```

Win	PM	Motif
Y	Y	I

W32s This member function has no effect.

Motif The User Interface Class Library supports only single threading; therefore, suspend and resume have no effect.

Exceptions	
IAccessError	The thread was not suspended. The thread identifier (ID) is invalid.

Thread Information

Use these members to query and set general thread information.

current Returns a reference to an object of the ICurrentThread (p. 243) class that represents the currently executing thread.

```
static ICurrentThread&
  current();
```

Win	PM	Motif
Y	Y	Y

currentHandle

Returns the thread handle for the current thread.

```
static IThreadHandle
  currentHandle();
```

Win	PM	Motif
Y	Y	I

PM The thread handle and thread ID are identical.

Win You must use the thread handle for the operating system thread and synchronization functions.

currentId Obtains the identifier (ID) of the currently executing thread.

```
static IThreadId
  currentId();
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Motif The AIX release of the User Interface Class Library only supports a single thread. Therefore, the current thread ID is always 1.

handle Returns the thread handle for the thread.

```
virtual IThreadHandle
  handle() const;
```

Win	PM	Motif
Y	Y	I

PM The thread handle and thread ID are identical.

Win You must use the thread handle for the operating system thread and synchronization functions.

id Obtains the identifier (ID) of the thread. A return value of 0 indicates that the thread is not started.

```
virtual IThreadId
  id() const;
```

Win	PM	Motif
Y	Y	Y

isStarted Determines if this thread is currently active. If it is currently active, true is returned.

```
virtual Boolean
  isStarted() const;
```

Win	PM	Motif
Y	Y	Y

setVariable Stores data and an associated key, on a per thread instance. You can later retrieve this data using IThread::variable (p. 996). You can store up to 16 keys and their associated data per thread.

```
IThread&
  setVariable( const IString& key,
               const IString& value );
```

Win	PM	Motif
Y	Y	Y

key Reference to an IString object that represents the search key that is associated with the stored data.

value Reference to an IString object that contains the data to be stored.

Exceptions	
IInvalidRequest	The keyed variable could not be set because the limit has been reached.

variable Returns the data associated with the *key* that was stored using IThread::setVariable (p. 996).

```
IString
  variable( const IString& key ) const;
```

Win	PM	Motif
Y	Y	Y

key Reference to an IString object that represents the search key.

Thread Priority

Use these members to control the thread's priority. You can query or set the priority class and level.

adjustPriority

Changes the thread's priority level by the specified amount.

```
virtual IThread&
  adjustPriority( int delta );
```

Win	PM	Motif
Y	Y	I

delta Integer value that represents the priority level delta. This value must be between -31 and 31.

PM The delta value is between -31 and 31.

W32s This member function has no effect.

Motif You cannot adjust the thread's priority on AIX.

Exceptions	
IAccessError	The thread priority was not adjusted. The priority *delta* may be invalid. The valid values are -31 to 31.

priorityClass

Obtains the priority class of this thread. The return value is an enumerator provided by IApplication::PriorityClass (p. 50).

```
virtual IApplication::PriorityClass
  priorityClass() const;
```

Win	PM	Motif
Y	Y	I

W32s This member function has no effect.

Motif The priority class for AIX is always IApplication::regular.

priorityLevel

Obtains the priority level of this thread. The return value is between 0 and 31.

```
virtual unsigned
  priorityLevel() const;
```

Win	PM	Motif
Y	Y	I

PM The value of the level is in the range 0-31.

© IBM Corp. 1992, 1996

W32s This member function always returns 0.

Motif The level for AIX is always 0.

setPriority Sets the priority class and level of this thread.

```
virtual IThread&
  setPriority( IApplication::PriorityClass priority,
               unsigned level );
```

Win	PM	Motif
Y	Y	I

priority IApplication::PriorityClass (p. 50) enumerator that identifies the priority class.

level Unsigned value that represents the priority level.

W32s This member function has no effect.

Motif You cannot control the priority of a thread in the AIX environment. This function has no effect on the priority of an executing thread.

Exceptions	
IAccessError	The thread priority was not set. The priority class or level may be invalid.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Constructors

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

operator = Assigns the member data of an object of this class to another object of this class, therefore preserving resource allocation.

```
IThread&
  operator =( const IThread& thread );
```

Win	PM	Motif
Y	Y	Y

thread Reference to an existing thread object.

Implementation

These members provide utilities used to implement this class. They are used by the User Interface Class Library and are not intended for use by users of this class.

newStartedThread

Creates an object of the IStartedThread class.

```
static IStartedThread*
  newStartedThread();
```

Win	PM	Motif
Y	Y	Y

startedThread

Returns a pointer to the object of the IStartedThread class that corresponds to this thread.

```
virtual IStartedThread*
  startedThread() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

IThread contains the following nested classes:

IThread::Cursor (see page 1001)

Nested Type Definitions

(unsigned long)

```
typedef void ( _System * SystemFnPtr ) ( unsigned long );
```

This typedef defines a pointer to a function that uses the _System calling convention.

© IBM Corp. 1992, 1996

Note: _System is the default calling convention for the OS/2 operating system.

(void *)

```
typedef void ( _Optlink * OptlinkFnPtr ) ( void * );
```

This typedef defines a pointer to a function that uses the _Optlink calling convention.

Note: _Optlink is the default calling convention for IBM VisualAge for C++.

IThread::Cursor

Derivation

IBase
  IVBase
    IThread::Cursor

Inherited By None.

Header File ithread.hpp

Members

Member	Page	Member	Page
Constructor	1001	setToFirst	1002
Cursor	1001	setToNext	1002
invalidate	1002	threadId	1002
isValid	1002	~Cursor	1002

The IThread::Cursor class creates and manages the cursor for an IThread (p. 981) object. This nested class defines objects that you can use to traverse or iterate through a set of active threads. In the same way that you can use a cursor to traverse the objects in a collection, you can use this cursor to traverse a set of threads one at a time.

Typically, you traverse a set, or collection, of active threads by doing the following:

1. Calling setToFirst (p. 1002)
2. Looping through the threads using setToNext (p. 1002)
3. Processing the returned thread IDs from threadId (p. 1002) until isValid (p. 1002) returns false

Note: This class only provides access to threads that an IThread (p. 981) object represents.

Public Functions

Constructors

Use these members to construct and destruct objects of this nested class.

Cursor Constructs objects of this class. This constructor accepts an optional flag that indicates how threads are enumerated.

© Copyright IBM Corp. 1992, 1996

IThread::Cursor

```
Cursor( Boolean allThreads = true );
```

Win	PM	Motif
Y	Y	Y

allThreads Boolean value that determines if all threads are enumerated or only those that have User Interface Class Library windows. The default is for all threads to be enumerated. Optional.

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	Y	Y

Thread Iteration

Use these members to iterate through the set of active threads.

invalidate Marks the cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

isValid Determines if the cursor is valid. If it is valid, true is returned.

```
Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

setToFirst Sets the cursor's position to the first thread.

```
Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

setToNext Advances the cursor's position to the next thread.

```
Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

threadId Obtains the identifier (ID) of the thread at the current cursor position.

```
IThreadId
  threadId() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The thread identifier (ID) was not returned. The cursor is not valid.

Inherited Public Functions

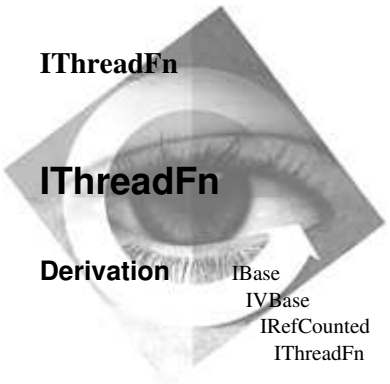
IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IThreadFn

Derivation

IBase
IVBase
IRefCounted
IThreadFn

Inherited By IThreadMemberFn

Header File ithread.hpp

Members

Member	Page
Constructor	1004
run	1005
~IThreadFn	1005

The IThreadFn class represents functions to be dispatched on secondary threads of execution when you start an IThread (p. 981) object. The User Interface Class Library reference-counts objects of this class to manage their destruction after the thread has terminated. The User Interface Class Library passes such a reference to IThread::start (p. 992), supplying the reference via the IReference (Vol. I) template class.

This class is an abstract thread function class.

Public Functions

Constructors

You cannot construct or destruct objects of this class because it is an abstract class. Use the template class IThreadMemberFn (p. 1010) for dispatching C++ member functions to an object on a new thread.

IThreadFn Provides the default and only constructor for this abstract class.

```
IThreadFn();
```

Win	PM	Motif
Y	*Y*	*Y*

© Copyright IBM Corp. 1992, 1996

~IThreadFn

```
virtual
  ~IThreadFn();
```

Win	PM	Motif
Y	Y	Y

Run Function

Use run function members to implement the code that you need to run on secondary threads of execution.

run Called when the thread function object is called. Override this function to implement the code that you need to run on secondary threads of execution.

```
virtual void
  run() = 0;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996

IThreadHandle

Derivation

IBase
IHandle
IThreadHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	1006
noHandle	1007

The IThreadHandle class is a wrapper for system-provided handles for threads.

PM An IThreadHandle is identical to an IThreadId. You can use them interchangeably.

Win Use IThreadHandle where a handle is needed, such as in the system thread and synchronization functions.

Public Functions

Constructors

You can construct objects of this class.

IThreadHandle

Constructs objects of this class from a thread handle (a value of the type IHandle::Value), which defaults to 0.

```
IThreadHandle( Value thread = 0 );
```

Win	PM	Motif
Y	*N*	*N*

Win If you are creating threads, use the return value from the CreateThread function to create a thread handle. You can also use IThread::currentHandle (p. 995) or ICurrentThread::handle (p. 243) to get the handle for the current thread.

© Copyright IBM Corp. 1992, 1996

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Public Data

Thread Handle Specifics

Contains thread-handle-specific values.

noHandle Represents the default value in the IThread (p. 982) constructor for the *threadHandle* parameter. This parameter indicates that the thread is already accessible to the User Interface Class Library or is the current thread.

```
static const IThreadHandle
  noHandle;
```

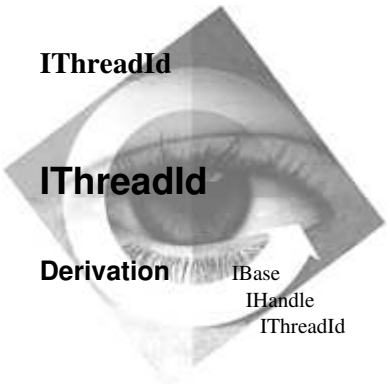
Win	PM	Motif
Y	*N*	*N*

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IThreadId

Derivation

IBase
IHandle
IThreadId

Inherited By None.

Header File ihandle.hpp

Members

Member	Page
Constructor	1008
operator Value	1008

The IThreadId class accesses numeric identifiers for threads.

PM IThreadId is an alias for the OS/2 Programmer's Toolkit type TID.

Motif In AIX, the User Interface Class Library does not support multiple threads. This class always returns a ThreadId of 1.

Public Functions

Constructors

You can construct objects of this class.

IThreadId Constructs objects of this class from a thread ID (a value of type IHandle::Value), which defaults to 0.

```
IThreadId( Value tid = 0 );
```

Win	PM	Motif
Y	Y	Y

Operators

This group contains operators for this class.

operator Value

Returns the IHandle value.

© Copyright IBM Corp. 1992, 1996

```
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**	

© IBM Corp. 1992, 1996

IThreadMemberFn

Derivation

IBase
IVBase
IRefCounted
IThreadFn
IThreadMemberFn

Inherited By None.

Header File ithread.hpp

Members

Member	Page
Constructor	1011
run	1011
~IThreadMemberFn	1011

The IThreadMemberFn template class is derived from IThreadFn (p. 1004) for dispatching C++ member functions to an object on a new thread. The template argument is the class of the object where the dispatched member functions are called.

The constructor for such objects requires the following:

- A reference to the object where the dispatched member functions are called
- A pointer to the member functions to call

The member functions return void and accept no parameters. You can add support for other member functions by deriving a class from this class.

Customization (Template Argument)

IThreadMemberFn is a template class that is created with the following template argument:

T The class of the object where the dispatched member functions are called.

Public Functions

Constructors

Use these members to construct and destruct objects of this template class.

© Copyright IBM Corp. 1992, 1996

IThreadMemberFn

You construct objects of this class by specifying an argument of the template argument class and a pointer to a member function of the template argument class.

```
IThreadMemberFn( T& obj,
                 void ( T::* mem ) ( ) );
```

Win	PM	Motif
Y	Y	Y

obj Object of the template argument class **T**.

mem Pointer to a member function of the template argument class. The member function should return a *void* parameter and accept no formal parameters.

~IThreadMemberFn

```
virtual
  ~IThreadMemberFn();
```

Win	PM	Motif
Y	Y	Y

Run Function

Use these members to run the member function that was specified when an object of this class was created.

run Calls the member function that you specify when you create an object of this class.

```
virtual void
  run();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IThreadFn		
run		

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

© IBM Corp. 1992, 1996

IThreadMemberFn

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ITimer

Derivation

IBase
  IVBase
    ITimer

Inherited By None.

Header File itimer.hpp

Members

Member	Page	Member	Page
Constructor	1014	operator ==	1013
asDebugInfo	1015	setInterval	1016
asString	1015	start	1017
id	1018	stop	1017
interval	1016	timerAt	1016
isStarted	1017	~ITimer	1015
operator =	1015		

The ITimer class creates and references periodic time-interval-based operations.

ITimer objects use timer identifier values above 1000. Limit your use of timer identifiers to values below 1000 when directly creating or managing timers using operating system calls.

The ITimer class uses the operating system message queue for timer expiration. This can cause timer actions to be delayed if the application is currently tying up the queue. If a timer event is delayed, the next timer event still occurs at the expected interval (there should be no timer drift).

Win Timer identifiers are limited to the value 0x7FFF on Windows. This allows portability to OS/2, where this limit is imposed by the operating system.

Public Functions

Comparison

Use these members to compare ITimer objects.

operator == Returns true if and only if the timers are identical.

© Copyright IBM Corp. 1992, 1996

```
Boolean
  operator ==( const ITimer timer );
```

Win	PM	Motif
Y	Y	N

timer A timer object.

Constructors

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

Note: The ITimer class does not support user-specified timer identifier values or timers that were not created with the ITimer class.

ITimer

1 `ITimer( const ITimer& timer );`

Win	PM	Motif
Y	Y	N

timer A reference to an existing timer object.

Constructs an ITimer object from an existing ITimer object.

2 `ITimer();`

Win	PM	Motif
Y	Y	N

Constructs an ITimer object using the default constructor.

Exceptions	
IOutOfSystemResource	The system is out of available timer identifiers.

3
```
ITimer( const IReference < ITimerFn >& timerFunction,
        unsigned long timerInterval = 1000 );
```

Win	PM	Motif
Y	Y	N

timerFunction
A reference to an object of the IReference template class that was created using a derived ITimerFn (p. 1021) class. You must dynamically allocate the object.

timerInterval
A value that represents the time interval. One second (1000 milliseconds) is the default.

Constructs an ITimer object with a specification of code to be run. Use this form of the constructor to construct a new ITimer and immediately start it. It is equivalent to using the default constructor and then calling the start function.

This constructor permits specification of a time interval to be used (in thousandths of a second).

This constructor is equivalent to using the default constructor and immediately calling ITimer::start (p. 1017).

4
```
ITimer( unsigned long timerIdentifier );
```

Win	PM	Motif
Y	Y	N

timerIdentifier
A value that represents the timer identifier (ID).

Constructs an ITimer object from the timer ID of an existing ITimer object.

Exceptions	
IInvalidParameter	The timer identifier value is not valid.

operator = Assigns an ITimer object to reference an existing timer.

```
ITimer&
  operator =( const ITimer& timer );
```

Win	PM	Motif
Y	Y	N

timer A reference to an existing timer object.

~ITimer

The destructor does not stop the timer or deallocate resources. If you want to stop the timer and deallocate resources, call the stop method to halt the started timer.

```
virtual
  ~ITimer();
```

Win	PM	Motif
Y	Y	N

Diagnostics

Use these members for diagnostic purposes. They return IString representation of the timer diagnostics.

asDebugInfo Returns a representation of the timer as diagnostic information.

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	N

asString Returns a string of the form “ITimer(ID)”, where *ID* is replaced by the actual identifier of the timer.

© IBM Corp. 1992, 1996

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	N

Expiration Interval

Use these members to access the timer expiration interval.

interval Returns the timer interval (in thousandths of a second).

```
unsigned long
  interval() const;
```

Win	PM	Motif
Y	Y	N

setInterval Sets the timer interval (in thousandths of a second). If no interval is set for a timer, then an interval of one second is used by default.

If this member function is called for a started timer, then the timer is stopped and restarted with the new interval. The ITimerFn (p. 1021) being used is saved and used when the timer is restarted.

An interval of 0 causes the timer to expire as quickly as possible.

```
virtual ITimer&
  setInterval( unsigned long aInterval = 1000 );
```

Win	PM	Motif
Y	Y	N

aInterval A value that represents the new timer interval.

PM When you run a timer on the OS/2 operating system, an effective minimum of about 1/18 of a second exists for timer intervals.

Retrieving Objects

Use these members to retrieve a timer object using ITimer::Cursor (p. 1019).

timerAt Returns an ITimer object at the position indicated by *cursor*.

```
static ITimer
  timerAt( const Cursor& cursor );
```

Win	PM	Motif
Y	Y	N

cursor A reference to an ITimer::Cursor object.

Exceptions	
IInvalidParameter	The ITimer::Cursor is invalid.

Starting and Stopping

Use these members to start and stop timers.

isStarted Returns true if the timer is currently active.

```
Boolean
  isStarted() const;
```

Win	PM	Motif
Y	Y	N

start Starts a time-interval-based operation with an arbitrary ITimerFn. The ITimerFn is passed in using an IReference so that the ITimerFn can be reference-counted and subsequently deleted when the timer is stopped.

The ITimerFn is called each time the timer interval expires.

1

```
virtual ITimer&
  start( const IReference < ITimerFn >& timerFunction );
```

Win	PM	Motif
Y	Y	N

timerFunction

A reference to an object of the IReference template class that was created using a derived ITimerFn (p. 1021) class. You must dynamically allocate the object.

Exceptions	
IInvalidRequest	The timer has already been started.
IAccessError	The operating system is unable to start the timer.

2

```
virtual ITimer&
  start( const IReference < ITimerFn >& timerFunction,
         unsigned long timerInterval );
```

Win	PM	Motif
Y	Y	N

timerFunction

A reference to an object of the IReference template class that was created using a derived ITimerFn (p. 1021) class. You must dynamically allocate the object.

timerInterval

A value that represents the timer interval.

stop Stops the timer. You must stop the timer to deallocate resources used by the timer.

```
virtual ITimer&
  stop();
```

Win	PM	Motif
Y	Y	N

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The operating system is unable to stop the timer.

Timer Information

Use these members to access timer information, such as the timer identifier.

id Returns the timer's identifier.

```
unsigned long
  id() const;
```

Win	PM	Motif
Y	*Y*	*N*

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

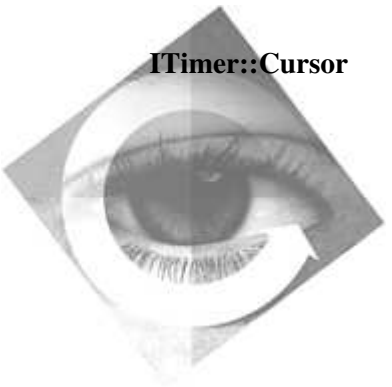
Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

Nested Classes

ITimer contains the following nested classes:

ITimer::Cursor (see page 1019)



ITimer::Cursor

Derivation

IBase
  IVBase
    ITimer::Cursor

Inherited By None.

Header File itimer.hpp

Members

Member	Page	Member	Page
Constructor	1019	setToFirst	1020
Cursor	1019	setToNext	1020
invalidate	1019	~Cursor	1019
isValid	1020		

The ITimer::Cursor nested class iterates over the active timers for the current application. Only timers that are started with the ITimer class are iterated over. Timers for all threads will be iterated.

Public Functions

Constructors

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

Cursor Construct objects of this class using this, the default constructor.

```
Cursor();
```

Win	PM	Motif
Y	*Y*	*N*

~Cursor

```
virtual
  ~Cursor();
```

Win	PM	Motif
Y	*Y*	*N*

Timer Iteration

Use these members to iterate through the set of active timers.

invalidate Invalidates the cursor.

© Copyright IBM Corp. 1992, 1996

ITimer::Cursor

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	N

isValid Returns true if the cursor is valid.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	N

setToFirst Initializes the cursor to the first timer. If no timers are active, then it returns false and invalidates the cursor.

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	N

setToNext Advances the cursor to the next timer. If no other timers are active, then it returns false and invalidates the cursor.

If you call setToNext with an invalid cursor, it sets the cursor to the first active timer.

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	N

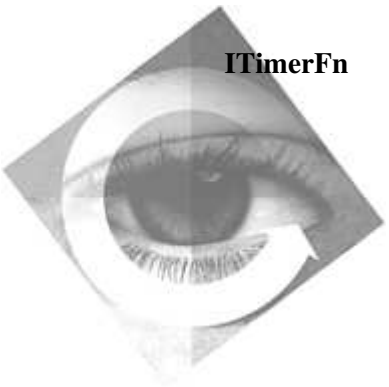
Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	



ITimerFn

Derivation

IBase
  IVBase
    IRefCounted
      ITimerFn

Inherited By

ITimerMemberFn
ITimerMemberFn0

Header File

itimer.hpp

Members

Member	Page
Constructor	1021
timerExpired	1022
~ITimerFn	1022

The ITimerFn class is an abstract timer function class. To use an ITimer (p. 1013) object, create a class derived from ITimerFn that implements the timerExpired (p. 1022) function. ITimerMemberFn (p. 1023) and ITimerMemberFn0 (p. 1026) are template classes that are derived from ITimerFn.

Objects of this class represent functions to be called when an active timer expires. These objects are reference-counted to manage their destruction (after the timer has been stopped). A reference to such an ITimerFn is passed to ITimer::start (p. 1017).

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign ITimerFn objects because both the copy constructor and the assignment operator are private members.

ITimerFn Construct objects of this class using this, the default constructor.

```
ITimerFn();
```

Win	PM	Motif
Y	*Y*	*N*

© Copyright IBM Corp. 1992, 1996

ITimerFn

~ITimerFn

```
virtual
  ~ITimerFn();
```

Win	PM	Motif
Y	Y	N

Timer Expiration

Use these members to perform actions each time the timer expires.

timerExpired Called each time the timer expires.

```
virtual void
  timerExpired( unsigned long timerId ) = 0;
```

Win	PM	Motif
Y	Y	N

timerId A value that represents the timer identifier (ID).

Inherited Public Functions

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

ITimerMemberFn

Derivation

IBase
  IVBase
    IRefCounted
      ITimerFn
        ITimerMemberFn

Inherited By None.

Header File itimer.hpp

Members

Member	Page
Constructor	1024
timerExpired	1024

The ITimerMemberFn template class is an ITimerFn-derived class for dispatching C++ member functions to an object when a timer expires. The template argument is as follows:

T The class of the object against which the member functions are applied.

Use this class instead of ITimerMemberFn0 (p. 1026) if you need the timer identifier (ID).

Objects of this class dispatch C++ member functions when a timer expires. The template argument is the class of the object for which the functions are called. The constructor for such objects requires a reference to the object that the member functions are to be applied to and a pointer to the member functions.

The member functions return void and accept a timer identifier (ID) argument.

Public Functions

Constructors

You can only construct objects of this class. You cannot copy or assign ITimerMemberFn objects because both the copy constructor and the assignment operator are private members.

© Copyright IBM Corp. 1992, 1996

ITimerMemberFn

Constructs objects of this class.

```
ITimerMemberFn(
   T& object,
   void ( T::* memberFunction ) ( unsigned long ) );
```

Win	PM	Motif
Y	*Y*	*N*

object A reference to an object of the template argument class **T**.

memberFunction
A pointer to a member function of the template argument class. The member function should return a *void* parameter and accept an *unsigned long* formal parameter that represents the timer ID.

Timer Expiration

Use these members to perform actions each time the timer expires.

timerExpired Calls the pointed-to member function that was provided at construction each time the timer expires.

```
virtual void
  timerExpired( unsigned long timerId );
```

Win	PM	Motif
Y	*Y*	*N*

timerId A value that represents the timer identifier (ID).

Inherited Public Functions

ITimerFn		
timerExpired		

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ITimerMemberFn0

Derivation

IBase
IVBase
IRefCounted
ITimerFn
ITimerMemberFn0

Inherited By None.

Header File itimer.hpp

Members

Member	Page
Constructor	1027
timerExpired	1027

The ITimerMemberFn0 template class is an ITimerFn-derived class for dispatching C++ member functions to an object when a timer expires. The template argument is as follows:

T The class of the object against which the member functions are applied.

Use this class instead of ITimerMemberFn (p. 1023) if you do not need the timer identifier (ID).

Objects of this class dispatch C++ member functions when a timer expires. The template argument is the class of the object for which the functions are called. The constructor for such objects requires a reference to the object that the member functions are to be applied to and a pointer to the member functions.

The member functions return void and accept no arguments.

Public Functions

Constructors

You can only construct objects of this class. You cannot copy or assign ITimerMemberFn0 objects because both the copy constructor and the assignment operator are private members.

© Copyright IBM Corp. 1992, 1996

ITimerMemberFn0

Constructs objects of this class.

```
ITimerMemberFn0( T& object,
                 void ( T::* memberFunction ) ( ) );
```

Win	PM	Motif
Y	*Y*	*N*

obj A reference to an object of the template argument class **T**.

memberFunction
A pointer to a member function of the template argument class. The member function should return a *void* parameter and accept no formal parameters.

Timer Expiration

Use these members to perform actions each time the timer expires.

timerExpired

Calls the pointed-to member function that was provided at construction each time the timer expires.

```
virtual void
  timerExpired( unsigned long timerId );
```

Win	PM	Motif
Y	*Y*	*N*

timerId A value that represents the timer identifier (ID).

Inherited Public Functions

ITimerFn		
timerExpired		

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

© IBM Corp. 1992, 1996

ITimerMemberFn0

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ITitle

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
ITitle

Inherited By None.

Header File ititle.hpp

Members

Member	Page	Member	Page
Constructor	1033	resetActiveTextBackgroundColor	1032
activeColor	1031	resetActiveTextForegroundColor	1032
activeTextBackgroundColor	1031	resetInactiveTextBackgroundColor	1032
activeTextBackgroundColorId	1038	resetInactiveTextForegroundColor	1032
activeTextForegroundColor	1031	setActiveTextBackgroundColor	1033
activeTextForegroundColorId	1039	setActiveTextForegroundColor	1033
borderColor	1031	setInactiveTextBackgroundColor	1033
enableNotification	1034	setInactiveTextForegroundColor	1033
handle	1030	setObjectText	1035
inactiveColor	1031	setText	1034
inactiveTextBackgroundColor	1032	setTitleText	1036
inactiveTextBackgroundColorId	1039	setViewNumber	1036
inactiveTextForegroundColor	1032	setViewText	1036
inactiveTextForegroundColorId	1039	text	1035
isValid	1030	textLength	1035
nativeRect	1037	viewNumber	1036
objectText	1035	viewNumberId	1039
objectTextId	1039	viewText	1037
owner	1030	viewTextId	1039
parentSize	1037	~ITitle	1034

The ITitle class creates and updates the title bar area of your frame window. The ITitle class consists of the following three components:

- Object text
- View text
- View number

When you construct a title, you must provide the object text, while the other two are optional. The User Interface Class Library separates the object text and view text

© Copyright IBM Corp. 1992, 1996

with a hyphen (-). The library separates the view text and view number with a colon (:). For example:

```
OS/2 System - Icon View:2
```

Use ITitle if you want the Common User Access (CUA) support for the object, view, and view number. If you do not, you can specify a title using an IFrameWindow constructor.

PM While a title may exceed 60 bytes in length, only the first 60 bytes appear in a Window List entry. See the function IFrameWindow::addToWindowList (p. 376) for information on adding entries to the window list.

Public Functions

Attributes

Use these members to query the accessible attributes of objects of this class.

handle Returns the window handle.

```
virtual IWindowHandle
  handle() const;
```

Win	PM	Motif
Y	*N*	*N*

isValid If this object represents a valid window in the window system, true is returned. If the window has yet to be created or has already been destroyed, false is returned.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	*N*	*N*

owner Returns the window's owner. This function may return 0 if either of the following occurs:

- The window was created without an owner
- The owner window is not represented by an IWindow object

Because a pointer value of 0 can cause unpredictable behavior, check the value of the returned pointer before using it.

```
virtual IWindow*
  owner() const;
```

Win	PM	Motif
Y	*N*	*N*

Colors

Use these members to query and set colors of the title area. You can query and set the active and inactive colors of the title area. You can query, set, and reset the active and inactive colors of the title's text foreground or background. You can query the border color of the title area.

activeColor Returns the active color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  activeColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default active color value of the title area.

activeTextBackgroundColor

Returns the active text background color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  activeTextBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default active text background color value of the title area.

activeTextForegroundColor

Returns the active text foreground color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  activeTextForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default active text foreground color value of the title area.

borderColor Returns the border color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  borderColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default border color value of the title area.

inactiveColor Returns the inactive color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  inactiveColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default inactive color value of the title area.

© IBM Corp. 1992, 1996

inactiveTextBackgroundColor

Returns the inactive text background color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  inactiveTextBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default inactive text background color value of the title area.

inactiveTextForegroundColor

Returns the inactive text foreground color value of the title area. If you have not set the color for the area, the default is returned.

```
virtual IColor
  inactiveTextForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default inactive text foreground color value of the title area.

resetActiveTextBackgroundColor

Resets the active text background color by undoing a previous set.

```
virtual ITitle&
  resetActiveTextBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetActiveTextForegroundColor

Resets the active text foreground color by undoing a previous set.

```
virtual ITitle&
  resetActiveTextForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetInactiveTextBackgroundColor

Resets the inactive text background color by undoing a previous set.

```
virtual ITitle&
  resetInactiveTextBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetInactiveTextForegroundColor

Resets the inactive text foreground color by undoing a previous set.

```
virtual ITitle&
  resetInactiveTextForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

setActiveTextBackgroundColor

Sets the active text background color to the specified color. The window area is identified by a system-defined presentation parameter value.

```
virtual ITitle&
  setActiveTextBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setActiveTextForegroundColor

Sets the active text foreground color to the specified color. The window area is identified by a system-defined presentation parameter value.

```
virtual ITitle&
  setActiveTextForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setInactiveTextBackgroundColor

Sets the inactive text background color to the specified color. The window area is identified by a system-defined presentation parameter value.

```
virtual ITitle&
  setInactiveTextBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setInactiveTextForegroundColor

Sets the inactive text foreground color to the specified color. The window area is identified by a system-defined presentation parameter value.

```
virtual ITitle&
  setInactiveTextForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

Constructors

You can construct and destruct objects of the ITitle class. You cannot copy or assign ITitle objects because both the copy constructor and the assignment operator are private functions.

ITitle

1

```
ITitle( IWindow* owner,
        const char* objectName = 0,
        const char* viewName = 0,
        unsigned long viewNumber = 0 );
```

Win	PM	Motif
Y	*Y*	*Y*

Use this function to construct ITitle objects from char*.

© IBM Corp. 1992, 1996

ITitle

Exceptions	
IInvalidRequest	The owner window of the ITitle object does not have a title bar. Make sure you have specified the correct owner window for the constructor.

2
```
ITitle( IWindow* owner,
        const IResourceId& objectName,
        const IResourceId& viewName = 0,
        unsigned long viewNumber = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this function to construct ITitle objects from IResourceIds of STRINGTABLE items.

Exceptions	
IInvalidRequest	The owner window of the ITitle object does not have a title bar. Make sure you have specified the correct owner window for the constructor.

~ITitle

```
virtual
  ~ITitle();
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

enableNotification

Enables the title to send notifications to any added observer objects.

```
virtual ITitle&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

Text Processing

Use these members to manage the title's text. You can query and set the text as well as query the length of the text of the title.

setText

Sets the title text.

1
```
virtual ITitle&
  setText( const char* text );
```

Win	PM	Motif
Y	N	Y

Use this function to set the title's text using a char*.

2
```
virtual ITitle&
  setText( const IResourceId& textResId );
```

Win	PM	Motif
Y	N	Y

Use this function to set the title's text using an IResourceId.

text Returns the title text.

```
virtual IString
  text() const;
```

Win	PM	Motif
Y	N	Y

textLength Returns the current length of the title text, in bytes.

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

Win	PM	Motif
Y	N	Y

Title Components

Use these members to query and set the title's components. You can set and query the text for the object or view. You can set and query the view number.

objectText Returns the object text.

```
virtual IString
  objectText() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The owner window of the ITitle object does not have a title bar. Make sure you have specified the correct owner window for the constructor.

setObjectText

Sets the object text.

Note: This function enforces the guideline that every window have a title. Therefore, you cannot use this function to put a null string in a frame window title. If you need to use a null string for a frame window title, call ITextControl::setText (p. 956) instead of this function.

1
```
virtual ITitle&
  setObjectText( const char* objectName );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the object text using a char*.

2
```
virtual ITitle&
  setObjectText( const IResourceId& objectNameResId );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

ITitle

Use this function to set the object text from a string table ID from a specified resource library.

setTitleText Sets all three components of a title (object text, view text, and view number) at once.

1

```
virtual ITitle&
  setTitleText( const char* objectName,
                const char* viewName = 0,
                unsigned long viewNum = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the object name and view name using character strings.

2

```
virtual ITitle&
  setTitleText( const IResourceId& objectNameResId,
                const IResourceId& viewNameResId = 0,
                unsigned long viewNum = 0 );
```

Win	PM	Motif
Y	Y	Y

Use this function to load the object and view text from a specified resource ID and resource library.

setViewNumber

Sets the view number.

```
virtual ITitle&
  setViewNumber( unsigned long viewNumber );
```

Win	PM	Motif
Y	Y	Y

setViewText Sets the view text.

1

```
virtual ITitle&
  setViewText( const IResourceId& viewNameResId );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the view text from a string table ID from a specified resource library.

2

```
virtual ITitle&
  setViewText( const char* viewName );
```

Win	PM	Motif
Y	Y	Y

Use this function to set the view text using a char*.

viewNumber Returns the view number.

```
unsigned long
  viewNumber() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The owner window of the ITitle object does not have a title bar. Make sure you have specified the correct owner window for the constructor.

viewText Returns the view text.

```
virtual IString
  viewText() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The owner window of the ITitle object does not have a title bar. Make sure you have specified the correct owner window for the constructor.

Window Positioning

Use these members to query the size and position of windows. Unless otherwise noted, the orientation of the coordinates accepted and returned by these members is the application orientation. For more information about coordinate orientation, see ICoordinateSystem (p. 230).

nativeRect Returns a rectangle representing the position and size of the window. Unlike IWindow::rect (p. 1079), this function always returns the position in the native GUI orientation.

Note: This function returns IRectangle(0,0,0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual IRectangle
  nativeRect() const;
```

Win	PM	Motif
Y	N	N

parentSize Returns an ISize object representing the size of the client rectangle in the parent window. The *client rectangle* is the coordinate space used by child windows to specify their location. If the current window or parent is determined to be the desktop, the size of the desktop is returned.

```
virtual ISize
  parentSize() const;
```

Win	PM	Motif
Y	N	N

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text

ITitle

ITextControl		
displaySize	setText	textLength

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 (p. 1044) are not shown.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (p. 1044) are not shown.

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

activeTextBackgroundColorId

Notification identifier provided to observers when the active text background color of the title changes.

```
static INotificationId const
  activeTextBackgroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

activeTextForegroundColorId

Notification identifier provided to observers when the active text foreground color of the title changes.

```
static INotificationId const
  activeTextForegroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

inactiveTextBackgroundColorId

Notification identifier provided to observers when the inactive text background color of the title changes.

```
static INotificationId const
  inactiveTextBackgroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

inactiveTextForegroundColorId

Notification identifier provided to observers when the inactive text foreground color of the title changes.

```
static INotificationId const
  inactiveTextForegroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

objectTextId

Notification identifier provided to observers when the object text of the title changes. ITitle provides a pointer to the new text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  objectTextId;
```

Win	PM	Motif
Y	*Y*	*Y*

viewNumberId

Notification identifier provided to observers when the view number of the title changes. ITitle provides the new view number in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  viewNumberId;
```

Win	PM	Motif
Y	*Y*	*Y*

viewTextId

Notification identifier provided to observers when the view text of the title changes. ITitle provides a pointer to the new text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

© IBM Corp. 1992, 1996

ITitle

```
static INotificationId const
  viewTextId;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Data

ITextControl		
textId		

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 (p. 1044) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ITitleNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler
        ITextControlNotifyHandler
          ITitleNotifyHandler

Inherited By None.

Header File ititlenh.hpp

Members

Member	Page
Constructor	1041
dispatchHandlerEvent	1042
~ITitleNotifyHandler	1041

The ITitleNotifyHandler class processes events for all classes of titles.

This class is designed to handle events that require the title class 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ITitleNotifyHandler

This is the default constructor and accepts no parameters.

```
ITitleNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

~ITitleNotifyHandler

```
virtual
  ~ITitleNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

ITitleNotifyHandler

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 the title observers if any of the following events are received:

- activeTextForegroundColor event
- activeTextBackgroundColor event
- inactiveTextForegroundColor event
- inactiveTextBackgroundColor event

```
virtual Boolean
  dispatchHandlerEvent( IEvent& anEvent );
```

Win	PM	Motif
Y	*Y*	*Y*

Inherited Protected Functions

ITextControlNotifyHandler		
dispatchHandlerEvent		

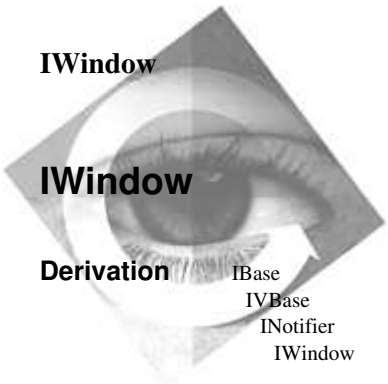
IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



IWindow

Derivation

IBase
IVBase
INotifier
IWindow

Inherited By

IControl
IFrameWindow
IHelpWindow
IMenu
IObjectWindow

Header File

iwindow.hpp

Members

Member	Page	Member	Page
Constructor	1058	disabledBackgroundColor	1052
acceleratorHandle	1047	disabledBackgroundColorId	1094
acceleratorTable	1047	disabledForegroundColor	1053
activeColor	1052	disabledForegroundColorId	1094
activeColorId	1093	disableMinimumSizeCaching	1064
addHandler	1085	disableNotification	1067
addObserver	1091	disableUpdate	1074
addRelatedHandleToWindowSet	1081	dispatch	1082
addToWindowSet	1082	dispatchRemainingHandlers	1061
asDebugInfo	1059	enable	1074
asString	1059	enableId	1094
backgroundColor	1052	enableMinimumSizeCaching	1064
backgroundColorId	1093	enableNotification	1067
borderColor	1052	enableUpdate	1075
borderColorId	1093	exceptionFunction	1062
calcMinimumSize	1091	extendedStyle	1087
capturePointer	1071	focusId	1094
characterSize	1063	font	1063
childAt	1052	fontId	1094
clipChildren	1096	foregroundColor	1053
clipSiblings	1097	foregroundColorId	1094
clipToParent	1097	handle	1049
color	1085	handleException	1062
commandId	1093	handleWithParent	1069
convertToGUIStyle	1074	handleWithPointerCaptured	1049
create	1086	hasFocus	1049
defaultOrdering	1073	hasPointerCaptured	1072
defaultProcedure	1082	helpId	1063
defaultPushButton	1064	hide	1075
deleteId	1093	hideSourceEmphasis	1075
deleteIsInProcess	1082	hiliteBackgroundColor	1053
desktopWindow	1049	hiliteBackgroundColorId	1095
disable	1074	hiliteForegroundColor	1053
disabled	1097	hiliteForegroundColorId	1095

© Copyright IBM Corp. 1992, 1996

IWindow

Member	Page	Member	Page
id	1049	removeFromWindowSet	1084
inactiveColor	1053	removeHandler	1088
inactiveColorId	1095	removeObserver	1092
isAutoDeleteObject	1067	removeRelatedHandleFromWindowSet	1084
isAutoDestroyWindow	1068	resetActiveColor	1054
isDragStarting	1081	resetBackgroundColor	1054
isEnabled	1074	resetBorderColor	1054
isEnabledForNotification	1067	resetColor	1088
isFrameWindow	1050	resetDisabledBackgroundColor	1054
isGroup	1050	resetDisabledForegroundColor	1054
isLayoutDistorted	1065	resetFont	1063
isMinimumSizeCachingEnabled	1065	resetForegroundColor	1054
isPrimaryWindow	1087	resetHiliteBackgroundColor	1054
isRelatedHandle	1080	resetHiliteForegroundColor	1055
isShowing	1075	resetInactiveColor	1055
isTabStop	1050	resetMinimumSize	1066
isValid	1050	resetShadowColor	1055
isVisible	1075	saveBits	1097
isWindowValid	1050	savedMinimumSize	1091
itemProvider	1059	saveMinimumSize	1091
layoutAdjustment	1065	sendEvent	1060
mapPoint	1078	setAcceleratorHandle	1048
matchForMnemonic	1065	setAcceleratorTable	1048
messageQueue	1051	setActiveColor	1055
minimumSize	1065	setAutoDeleteObject	1068
movePointerTo	1072	setAutoDestroyWindow	1068
moveSizeTo	1078	setBackgroundColor	1055
moveTo	1078	setBorderColor	1056
nativeRect	1078	setColor	1088
noStyle	1097	setDefaultOrdering	1073
notificationHandler	1091	setDisabledBackgroundColor	1056
notifyObservers	1068	setDisabledForegroundColor	1056
objectWindow	1051	setExceptionFunction	1062
observerList	1092	setExtendedStyle	1088
owner	1069	setFocus	1051
parent	1069	setFont	1063
parentSize	1079	setForegroundColor	1056
passEventToOwner	1083	setHelpId	1064
pointerPosition	1072	setHiliteBackgroundColor	1056
position	1079	setHiliteForegroundColor	1056
positionBehindSibling	1073	setId	1051
positionBehindSiblings	1073	setInactiveColor	1057
positionId	1095	setItemProvider	1059
positionOnSiblings	1073	setLayoutDistorted	1066
postEvent	1059	setMinimumSize	1066
prepareForUse	1087	setNotificationHandler	1092
presSpace	1076	setOwner	1070
rect	1079	setParent	1070
refresh	1076	setShadowColor	1057
registerCallbacks	1083	setStyle	1089
releasePointer	1072	setWindowData	1089
releasePresSpace	1077	shadowColor	1057
removeAllObservers	1092	shadowColorId	1095

© IBM Corp. 1992, 1996

IWindow

Member	Page	Member	Page
show	1077	unregisterCallbacks	1085
showSourceEmphasis	1077	visible	1097
size	1079	visibleId	1096
sizeId	1096	visibleRectangle	1067
sizeTo	1079	windowULong	1090
startHandlingEventsFor	1084	windowUShort	1090
style	1090	windowWithHandle	1051
synchPaint	1097	windowWithOwner	1070
systemCommandId	1096	windowWithParent	1071
topHandle	1057	~IWindow	1058

The IWindow class is the base window class and provides behavior common to all windows. Although this class contains behavior requiring the existence of a presentation window, you must construct the presentation window itself using a derived class.

Note: Although you can construct objects of this class directly, you do not generally do so.

In Motif, the User Interface Class Library does not support dialog templates. However, there is still an IWindow constructor that takes a parent IWindow and a resource ID. This is useful in case there is an existing widget you want to create an IWindow for, but you only know the following:

- It is a child of a known window.
- Its widget name, which the constructor uses as the resource ID.

IWindow wraps and enhances the X Windows System and Motif widget set. The implementation uses calls to the Motif toolkit (Xt...) and Motif helper (Xm...) routines, as well as a few calls directly to the X Library (XLib...) when Xt does not offer the necessary function.

This class is portable across the OS/2, AIX, and Windows environments. Most of the member functions are also portable. See individual member functions for details.

Public Functions

Accelerator Key Support

Accelerator keys are shortcut keys for performing an action. The action can be an application command, system command, or request for help. You can assign accelerator keys to any window.

If you press an accelerator key used by the window with the input focus, that window is sent the resulting command or help request. If the window does not translate the key into a command or

help request, a window in the parent window chain (up to and including the first frame window) could translate the key. In this case, the parent window that has the key in its accelerator table receives a command event, or the window with the input focus receives a help request.

acceleratorHandle

Returns a handle for the accelerator keys used by the window. If the window has no accelerator keys, this function returns a zero IAccelTblHandle (p. 42) object. Accelerator keys provided by the system (for example, F10 and Alt+F4) are not reflected in the returned object.

You can define the accelerator keys for a window using any of the following means:

- Construct an IFrameWindow (p. 349) object using the style IFrameWindow::accelerator (p. 383)
- Call setAcceleratorTable (p. 1048)
- Modify the contents of an IAcceleratorTable (p. 30) object constructed for the window
- Call setAcceleratorHandle (p. 1048)
- Construct an IAccelerator (p. 17) object for the window

```
IAccelTblHandle
  acceleratorHandle() const;
```

Win	PM	Motif
Y	*Y*	*N*

acceleratorTable

Returns the accelerator keys used by the window. If the window has no accelerator keys, this function returns an IAcceleratorTable (p. 30) object that contains no keys. Accelerator keys provided by the system (for example, F10 and Alt+F4) are not included in the returned object.

Calling this function is equivalent to using the IAcceleratorTable constructor (p. 34) that accepts a const IWindow* parameter.

You can define the accelerator keys for a window using any of the following means:

- Construct an IFrameWindow (p. 349) object using the style IFrameWindow::accelerator (p. 383)
- Call setAcceleratorTable (p. 1048)
- Modify the contents of an IAcceleratorTable object constructed for the window
- Call setAcceleratorHandle (p. 1048)
- Construct an IAccelerator (p. 17) object for the window

© IBM Corp. 1992, 1996

```
IAcceleratorTable
  acceleratorTable() const;
```

Win	PM	Motif
Y	*Y*	*N*

setAcceleratorHandle

Replaces the accelerator keys used by the window. Specify a zero IAccelTblHandle (p. 42) object to remove the window's accelerator keys.

Calling this function is equivalent to any of the following:

- Calling setAcceleratorTable (p. 1048)
- Changing an IAcceleratorTable (p. 30) object constructed for the window
- Constructing an IFrameWindow (p. 349) object with the style IFrameWindow::accelerator (p. 383)
- Constructing an IAccelerator (p. 17) object for the window

```
IWindow&
  setAcceleratorHandle( const IAccelTblHandle& handle );
```

Win	PM	Motif
Y	*Y*	*N*

setAcceleratorTable

Replaces the accelerator keys used by the window. Specify a zero pointer or an empty IAcceleratorTable (p. 30) object to remove the window's accelerator keys.

Calling this function is equivalent to any of the following:

- Calling setAcceleratorHandle (p. 1048)
- Changing an IAcceleratorTable object constructed for the window
- Constructing an IFrameWindow (p. 349) object with the style IFrameWindow::accelerator (p. 383)
- Constructing an IAccelerator (p. 17) object for the window

```
IWindow&
  setAcceleratorTable(
   const IAcceleratorTable* acceleratorTable);
```

Win	PM	Motif
Y	*Y*	*N*

Attributes

Use these members to query and set the accessible attributes of objects of this class.

desktopWindow

Returns the object representing the system desktop window.

```
static IWindow*
  desktopWindow();
```

Win	PM	Motif
Y	Y	Y

Motif Returns the User Interface Class Library object representing the X root window. In other window systems, the desktop is the root of all windows and its handle can be obtained and used. In X-Motif, the user does not know the actual handle of the root window. Instead, the library provides the user a set of routines for creating children of the root. These children cannot be visible widgets: they are only invisible shell widgets. Therefore, they are used as the parent of the actual visible widgets you want to hang off the root. The User Interface Class Library hides this difference from you if you use IWindow objects, including derived classes, and not window handles.

handle

Returns the window handle.

```
virtual IWindowHandle
  handle() const;
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidRequest	The window does not have a valid handle.

handleWithPointerCaptured

Returns the window handle that currently has the mouse captured.

```
static IWindowHandle
  handleWithPointerCaptured();
```

Win	PM	Motif
Y	Y	N

hasFocus

If the window has the input focus, true is returned. Otherwise, false is returned.

```
virtual Boolean
  hasFocus() const;
```

Win	PM	Motif
Y	Y	Y

id

Returns the window identifier of the window.

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

Win	PM	Motif
Y	Y	Y

Motif Motif's equivalent of window IDs are widget names. The User Interface Class Library has established the convention of numeric IDs for portability. The library converts the numeric to a character string and uses the string as the name of the

© IBM Corp. 1992, 1996

widget on XmCreate... By doing so, the library can extract the name from the widget instead of keeping a local copy of the ID:

```
IString resid( XtName( (Widget)handle() );
return ( resid.asInt() );
```

In Motif, the User Interface Class Library classes are the only classes that use the IWindow::id value as an integer.

If this IWindow was constructed from an existing widget, the widget name is typically a string of any alphanumeric characters, not just 0-9. In this case, the return value is undefined. If you use the IWindowHandle constructor for IWindow or any of its derived classes and you did not adhere to the User Interface Class Library convention of numeric window IDs, do not use this member function. Instead, use XtNameToWidget.

Note: Using IResourceId objects instead of the primitives for int or char* ensure adherence to the convention.

isFrameWindow

If this object represents a frame window, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isFrameWindow() const;
```

Win	PM	Motif
Y	Y	Y

isGroup

If the control has the group style set, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isGroup() const;
```

Win	PM	Motif
Y	Y	Y

isTabStop

If the user can tab to the window, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isTabStop() const;
```

Win	PM	Motif
Y	Y	Y

isValid

If this object represents a valid window in the window system, true is returned. If the window has yet to be created or has already been destroyed, false is returned.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

isWindowValid

If the specified pointer is the address of an IWindow object managed by the User Interface Class Library, true is returned. Otherwise, false is returned.

```
static Boolean
  isWindowValid( const IWindow* window );
```

Win	PM	Motif
Y	Y	N

messageQueue

Returns the handle for the window's message queue.

```
IMessageQueueHandle
  messageQueue() const;
```

Win	PM	Motif
N	Y	N

objectWindow

Returns the object representing the system object window.

```
static IWindow*
  objectWindow();
```

Win	PM	Motif
Y	Y	Y

Motif The object window is the same as the desktop window when on X-Motif systems. In X, with both widgets destined to be visible and purely logical, nonvisual widgets are in the same widget hierarchy, freely intermixed. Other systems have distinct hierarchies for these two kinds of objects, and so the User Interface Class Library has two objects to ensure portability.

setFocus

Sets the input focus to the window.

```
virtual IWindow&
  setFocus();
```

Win	PM	Motif
Y	Y	Y

setId

Sets the window identifier of the window.

```
virtual IWindow&
  setId( unsigned long newIdentifier );
```

Win	PM	Motif
Y	Y	I

windowWithHandle

Returns the IWindow object for the specified window handle. If the specified window handle is not associated with any IWindow object, 0 is returned. If *allThreads* is set to true, all threads are searched. Otherwise, only the current thread is searched. This is a static function.

```
static IWindow*
  windowWithHandle( const IWindowHandle& windowHandle,
                    Boolean allThreads = true );
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Child Cursor Support

These members provide child cursor support.

childAt Returns the window handle of the child window at the current position of the specified cursor.

```
IWindowHandle
  childAt( const ChildCursor& cursor ) const;
```

Win	PM	Motif
Y	Y	Y

Colors

Use these members to query, set, and reset colors for IWindow classes and their derived classes.

activeColor Returns the active color value of the window area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  activeColor() const;
```

Win	PM	Motif
I	Y	I

Win Returns the default active color value of the window area.

backgroundColor

Returns the background color value of the window. If you have not set the background color for the window, the default background color is returned.

```
virtual IColor
  backgroundColor() const;
```

Win	PM	Motif
Y	Y	Y

borderColor Returns the border color value of the window area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  borderColor() const;
```

Win	PM	Motif
I	Y	Y

Win Returns the default border color value of the window area.

disabledBackgroundColor

Returns the disabled background color value of the window area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  disabledBackgroundColor() const;
```

Win	PM	Motif
I	Y	I

Win Returns the default disabled background color value of the window area.

disabledForegroundColor

Returns the disabled foreground color value of the window area. If you have not set the color for the area, the default color is returned.

```
virtual IColor
  disabledForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default disabled foreground color value of the window area.

foregroundColor

Returns the foreground color value of the window or the default if no foreground color for the window has been set.

```
virtual IColor
  foregroundColor() const;
```

Win	PM	Motif
Y	*Y*	*Y*

hiliteBackgroundColor

Returns the highlight background color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  hiliteBackgroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default highlight background color value of the window area.

hiliteForegroundColor

Returns the highlight foreground color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  hiliteForegroundColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default highlight foreground color value of the window area.

inactiveColor

Returns the inactive color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  inactiveColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Win Returns the default inactive color value of the window area.

© IBM Corp. 1992, 1996

IWindow

resetActiveColor

Resets the active color by undoing the previous set.

```
virtual IWindow&
  resetActiveColor();
```

Win	PM	Motif
I	*Y*	*I*

resetBackgroundColor

Resets the background color by undoing a previous set.

```
virtual IWindow&
  resetBackgroundColor();
```

Win	PM	Motif
Y	*Y*	*I*

resetBorderColor

Resets the border color by undoing a previous set.

```
virtual IWindow&
  resetBorderColor();
```

Win	PM	Motif
I	*Y*	*I*

resetDisabledBackgroundColor

Resets the disabled background color by undoing a previous set.

```
virtual IWindow&
  resetDisabledBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetDisabledForegroundColor

Resets the disabled foreground color by undoing a previous set.

```
virtual IWindow&
  resetDisabledForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetForegroundColor

Resets the foreground color by undoing a previous set.

```
virtual IWindow&
  resetForegroundColor();
```

Win	PM	Motif
Y	*Y*	*I*

resetHiliteBackgroundColor

Resets the highlight background color by undoing a previous set.

```
virtual IWindow&
  resetHiliteBackgroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetHiliteForegroundColor

Resets the highlight foreground color by undoing a previous set.

```
virtual IWindow&
  resetHiliteForegroundColor();
```

Win	PM	Motif
I	*Y*	*I*

resetInactiveColor

Resets the inactive color by undoing a previous set.

```
virtual IWindow&
  resetInactiveColor();
```

Win	PM	Motif
I	*Y*	*I*

resetShadowColor

Resets the shadow color by undoing a previous set.

```
virtual IWindow&
  resetShadowColor();
```

Win	PM	Motif
I	*Y*	*I*

setActiveColor

Sets the active color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setActiveColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setBackgroundColor

Sets the background color to the indicated color.

```
virtual IWindow&
  setBackgroundColor( const IColor& color );
```

Win	PM	Motif
Y	*Y*	*Y*

Motif The Motif version of this function uses XmChangeColor to set the background color. XmChangeColor also sets the shadow and foreground colors to various shades of the background color to achieve a three-dimensional look for the widget. If you want the foreground color to be a different color than the color set by XmChangeColor, set the foreground color after setting the background color using IWindow::setForegroundColor.

© IBM Corp. 1992, 1996

IWindow

setBorderColor

Sets the border color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setBorderColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*Y*

setDisabledBackgroundColor

Sets the disabled background color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setDisabledBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setDisabledForegroundColor

Sets the disabled foreground color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setDisabledForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setForegroundColor

Sets the foreground color to the indicated color.

```
virtual IWindow&
  setForegroundColor( const IColor& color );
```

Win	PM	Motif
Y	*Y*	*Y*

setHiliteBackgroundColor

Sets the highlight background color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setHiliteBackgroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setHiliteForegroundColor

Sets the highlight foreground color to the indicated color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setHiliteForegroundColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setInactiveColor

Sets the inactive color to the specified color. The window area is identified by a system-defined presentation parameter value.

```
virtual IWindow&
  setInactiveColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

setShadowColor

Resets the shadow color by undoing a previous set.

```
virtual IWindow&
  setShadowColor( const IColor& color );
```

Win	PM	Motif
I	*Y*	*I*

shadowColor

Returns the shadow color value of the window area or the default if no color for the area has been set.

```
virtual IColor
  shadowColor() const;
```

Win	PM	Motif
I	*Y*	*I*

Compound Control

Use these members to access the top window system. A *compound control* is a window that consists of more than one window system control.

topHandle

Returns the top (that is, the oldest ancestor) window system control of the controls created by this class in its constructor or elsewhere. The User Interface Class Library internally uses the returned handle for, among other things, showing this object.

```
virtual IWindowHandle
  topHandle() const;
```

Win	PM	Motif
N	*N*	*Y*

Constructors

You can construct and destruct objects of the IWindow class. You cannot copy or assign IWindow objects because both the copy constructor and the assignment operator are private functions.

© IBM Corp. 1992, 1996

IWindow

1 `IWindow( unsigned long identifier,`
`IWindow* parent );`

Win	PM	Motif
Y	Y	Y

Constructs an IWindow object for an existing window that is a known child of a known IWindow object. For example, when a dialog template is loaded, you can use this function to create an IWindow object for one of its child controls; you must know the window identifier of the child control.

Motif This method is seldom used in Motif because dialog-template loading is not supported.

2 `IWindow( const IWindowHandle& handle );`

Win	PM	Motif
Y	Y	Y

Constructs an IWindow object for an existing window, typically created by having previously called a window system API (for example, WinCreateWindow(...), XtCreateWidget(...)). The window is not automatically destroyed when the IWindow object is destroyed; you can change this by using setAutoDestroyWindow(true).

Exceptions	
IInvalidParameter	The specified *handle* is invalid. You must specify the handle of an existing window.

~IWindow

Cleans up as this IWindow object is being destroyed. Clean up includes the following:

- Cueing any canvas objects containing this object to adjust their layout
- Destroying the associated window for this object if the autoDestroyWindow flag is true.

```
virtual
  ~IWindow();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system was unable to reset the default window procedure; a possible cause is an invalid window handle.
IInvalidRequest	An attempt was made to destroy a window from within a handler which was handling an event for the same window. To avoid this situation, delete the object from outside the handler or use setAutoDeleteObject (p. 1068) to cause the library to automatically delete the object.

Diagnostics

Use these members to output diagnostic information for the window.

asDebugInfo Returns a string containing detailed information about the window, including the information provided by IWindow::asString (p. 1059).

```
virtual IString
  asDebugInfo() const;
```

Win	PM	Motif
Y	Y	Y

asString Returns a string containing the window handle and ID.

```
virtual IString
  asString() const;
```

Win	PM	Motif
Y	Y	Y

Drag and Drop Support

Use these members to query and set the drag-item provider for objects of this class.

itemProvider Returns a pointer to the drag-item provider (IDMItemProvider) for this window.

```
IDMItemProvider*
  itemProvider() const;
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function.

setItemProvider

Sets a pointer to the drag-item provider (IDMItemProvider) for this window.

```
IWindow&
  setItemProvider( IDMItemProvider* dragProvider );
```

Win	PM	Motif
Y	Y	Y

Motif AIX does not support this function.

Event Send and Post

Use these members to send or post an event to a window.

postEvent Posts an event constructed from the arguments to the window. If you specify an enumerator from IWindow::EventType (p. 1098), the library constructs an event with the appropriate message identifier.

1
```
virtual const IWindow&
  postEvent( const IEvent& event ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) posted using this member function are subject to the restrictions and limitations of the Win32s system.

© IBM Corp. 1992, 1996

For example, when posting an event with a user defined eventId (or message), the high word of the first parameter is lost.

Events (or messages) posted using this member function are subject to the restrictions and limitations of the underlying operating system.

2
```
virtual const IWindow&
  postEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) posted using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when posting a user defined eventId (or message), the high word of the first parameter is lost.

3
```
virtual const IWindow&
  postEvent( EventType eventType,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

Events (or messages) posted using this member function are subject to the restrictions and limitations of the underlying operating system since the User Interface Class Library converts IWindow::EventType (p. 1098) enum values to system defined event identifiers.

sendEvent Sends an event constructed from the arguments to the window. If you specify an enumerator from IWindow::EventType (p. 1098), the library constructs an event with the appropriate message identifier.

1
```
virtual IEventResult
  sendEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) sent using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when sending a user defined eventId (or message), the high word of the first parameter is lost.

2
```
virtual IEventResult
  sendEvent( const IEvent& event ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) sent using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when sending an event with a user defined eventId (or message), the high word of the first parameter is lost.

3

```
virtual IEventResult
  sendEvent( EventType eventType,
             const IEventParameter1& parm1 = 0,
             const IEventParameter2& parm2 = 0 ) const;
```

Win	PM	Motif
Y	Y	Y

Events (or messages) posted using this member function are subject to the restrictions and limitations of the underlying operating system since the User Interface Class Library converts IWindow::EventType (p. 1098) enum values to system defined event identifiers.

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

dispatchRemainingHandlers

Dispatches the event to all handlers that the event has not yet been dispatched to. This function stops dispatching if any handler it calls returns true. If all remaining handlers return false, this function optionally calls IHandler::defaultProcedure (p. 415) depending on the value of the callDefProc parameter.

When this function is called, the event passed to it must be the same event that was passed to the handler you are calling this function from or an event constructed from that event.

If false is returned from this function, the caller must take one of the following actions:

- Handle the event and return true
- Call IHandler::defaultProcedure (p. 415) and return true

The caller of this function must always return true from the handler this function was called from.

If this function returns true, it means that either a handler the event was dispatched to handled the event, or none of the handlers handled the event and IHandler::defaultProcedure (p. 415) was called. If false is returned, it means that all handlers called returned false and did not handle the event and IHandler::defaultProcedure (p. 415) was not called.

© IBM Corp. 1992, 1996

```
Boolean
  dispatchRemainingHandlers( IEvent& event,
                             Boolean callDefProc = true );
```

Win	PM	Motif
Y	Y	N

Exceptions	
IInvalidRequest	The window dispatcher found a handler that was invalid. The handler must be removed prior to deletion.

Exception Processing

Use these members to process C++ exceptions in the IWindow dispatcher.

exceptionFunction

Returns a pointer to the current exception function. If one has not been registered, 0 is returned.

```
static IWindow::ExceptionFn*
  exceptionFunction();
```

Win	PM	Motif
Y	Y	Y

handleException

Called when the library detects an exception during the dispatch of an event. The default behavior determines if an IWindow::ExceptionFn (p. 1103) object has been registered and calls it. Otherwise, false is returned and the exception is thrown again.

```
virtual Boolean
  handleException( IException& dispatcherException,
                   IEvent& exceptionEvent );
```

Win	PM	Motif
Y	Y	Y

setExceptionFunction

Stores an IWindow::ExceptionFn (p. 1103) object to be called when an uncaught exception is detected while dispatching window events. If an object has already been registered, the IWindow::ExceptionFn is returned. Otherwise, 0 is returned.

```
static IWindow::ExceptionFn*
  setExceptionFunction(
   IWindow::ExceptionFn* exceptionFunction);
```

Win	PM	Motif
Y	Y	Y

Fonts

Use these members to set, reset, and query fonts.

characterSize

Calculates and returns the average character width and maximum character height for the currently set font.

```
ISize
  characterSize() const;
```

Win	PM	Motif
Y	Y	Y

font Returns the font used by the window.

```
virtual IFont
  font() const;
```

Win	PM	Motif
Y	Y	Y

resetFont Causes the window to disregard a font set by a call to setFont (p. 1063). Following a call to this function, the window uses a default font.

```
virtual IWindow&
  resetFont();
```

Win	PM	Motif
Y	Y	N

PM If a font has been set in the owner window chain, the window inherits the font used by its owner window. Otherwise, the window uses the default system font.

setFont Sets a new font to be used by the window.

```
virtual IWindow&
  setFont( const IFont& font );
```

Win	PM	Motif
Y	Y	Y

Help Support

Information about using your application can be displayed in help windows using the IHelpWindow (p. 442) class. Each topic of help information is displayed in a help panel that is identified by a number, help ID, that is unique to that application. You can associate each IWindow object in your application with a help panel by either providing a help table to your IHelpWindow object, see IHelpWindow::setHelpTable (p. 449), or by storing the help panel identifier with each IWindow object.

These functions allow you to set and query the help identifier for this IWindow object.

helpId Returns the identifier of the help panel that is to be displayed when help is requested for the window. To display the help panel, you must first associate the frame window containing this window with a help instance. See the IHelpWindow (p. 442) class for more information.

```
unsigned long
  helpId() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

setHelpId

Stores the identifier of the help panel that is to be displayed when help is requested for the window. To display the help panel, you must first associate the frame window containing this window with a help instance. See the IHelpWindow (p. 442) class for more information.

```
IWindow&
  setHelpId( unsigned long helpTopicId );
```

Win	PM	Motif
Y	Y	Y

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior. The virtual functions contain minimal implementations, which can be overridden in derived classes.

defaultPushButton

Returns the first child window that is a default push button, if one exists.

```
virtual IWindowHandle
  defaultPushButton() const;
```

Win	PM	Motif
Y	Y	Y

disableMinimumSizeCaching

Disables the caching of minimum sizes. When you disable caching, you cause each call to IWindow::minimumSize (p. 1065) to call calcMinimumSize (p. 1091).

By default, a window caches its minimum sizes.

```
IWindow&
  disableMinimumSizeCaching();
```

Win	PM	Motif
Y	Y	N

enableMinimumSizeCaching

Enables or disables the caching of minimum sizes as an optimization for not calling calcMinimumSize (p. 1091). When enabled, this optimization causes IWindow::minimumSize (p. 1065) to call calcMinimumSize, but thereafter to return the previously calculated value. When you disable caching, you cause each call to IWindow::minimumSize to call calcMinimumSize. See IWindow::minimumSize for details.

By default, a window caches its minimum sizes.

```
IWindow&
  enableMinimumSizeCaching( Boolean enableCaching = true );
```

Win	PM	Motif
Y	Y	N

isLayoutDistorted

If changes have been made in a window that requires updating the layout of the window in a canvas, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isLayoutDistorted( unsigned long layoutAttribute ) const;
```

Win	PM	Motif
Y	Y	Y

isMinimumSizeCachingEnabled

Returns if the IWindow object will call IWindow::savedMinimumSize (p. 1091) as an optimization for not calling calcMinimumSize (p. 1091). By default, a window uses this optimization.

```
Boolean
  isMinimumSizeCachingEnabled() const;
```

Win	PM	Motif
Y	Y	N

layoutAdjustment

The *layout adjustment* is the dimensions that a window is moved or sized or both after a canvas runs its layout routines. This function exists to support controls such as the drop-down combination box where the displayed size of the control is different from its actual size. The default behavior of this function returns IRectangle(0,0,0,0).

```
virtual IRectangle
  layoutAdjustment() const;
```

Win	PM	Motif
Y	Y	Y

matchForMnemonic

Returns the first child window using the specified character as a mnemonic.

```
virtual IWindowHandle
  matchForMnemonic( unsigned short character ) const;
```

Win	PM	Motif
Y	Y	Y

minimumSize

Returns the minimum allowable size that is set by a derived class. If setMinimumSize (p. 1066) has not been called or *windowCalculatedSize* is true, it returns the value of calcMinimumSize (p. 1091) or savedMinimumSize (p. 1091). Otherwise, it returns the value specified by setMinimumSize.

Presentation Manager and Windows: Because calcMinimumSize can be a potentially time-consuming function, IWindow::minimumSize will save the value returned by that function and return the saved value until the window has a new minimum size. This function saves the minimum size using saveMinimumSize (p. 1091) and then calls setLayoutDistorted (p. 1066) to clear the window's

© IBM Corp. 1992, 1996

minimumSizeChanged (p. 1099) flag. It retrieves the saved value using savedMinimumSize (p. 1091). It determines if the window has a new minimum size by checking if any of the following are true:

- setMinimumSize has been called
- isLayoutDistorted (p. 1065) indicates the window's minimumSizeChanged flag has been set
- savedMinimumSize returns a zero size

If the window's minimum size has changed, this function returns the minimum size set by setMinimumSize or returned by calcMinimumSize.

You can disable this optimization using disableMinimumSizeCaching (p. 1064).

```
ISize
  minimumSize( Boolean windowCalculatedSize = false ) const;
```

Win	PM	Motif
Y	Y	Y

resetMinimumSize

Resets the minimum allowable size as if setMinimumSize (p. 1066) had not been called.

```
IWindow&
  resetMinimumSize();
```

Win	PM	Motif
Y	Y	Y

setLayoutDistorted

Indicates that changes have occurred in the window causing the layout of the window in a canvas to be updated.

```
virtual IWindow&
  setLayoutDistorted( unsigned long layoutAttributesOn,
                      unsigned long layoutAttributesOff );
```

Win	PM	Motif
Y	Y	Y

setMinimumSize

Sets the minimum allowable size of the window. In the User Interface Class Library this applies only to windows on a canvas. Calling this function causes minimumSize (p. 1065) to not call calcMinimumSize (p. 1091).

```
IWindow&
  setMinimumSize( const ISize& size );
```

Win	PM	Motif
Y	Y	Y

visibleRectangle

Returns the painted rectangle for controls like the drop-down combination box where the painted rectangle of the control is different from its actual rectangle. The default behavior of this function returns the actual rectangle.

```
virtual IRectangle
  visibleRectangle() const;
```

Win	PM	Motif
Y	Y	Y

Notification Members

Use these members to identify and enable notifications sent to observer objects.

disableNotification

Causes the window to stop sending notifications to all added observer objects.

```
virtual IWindow&
  disableNotification();
```

Win	PM	Motif
Y	Y	Y

enableNotification

Enables or disables the window to send notifications to any added observer objects.

```
virtual IWindow&
  enableNotification( Boolean enable = true );
```

Win	PM	Motif
Y	Y	Y

isEnabledForNotification

If the window is sending notifications to observer objects, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isEnabledForNotification() const;
```

Win	PM	Motif
Y	Y	Y

Object Deletion

Use these members to manage the destruction of the window object.

isAutoDeleteObject

If the window object is deleted when a destroy event is dispatched to the window, true is returned. Otherwise, false is returned.

© IBM Corp. 1992, 1996

```
Boolean
  isAutoDeleteObject() const;
```

Win	PM	Motif
Y	Y	Y

isAutoDestroyWindow

If the presentation window is destroyed when the window object is deleted, true is returned. Otherwise, false is returned.

```
Boolean
  isAutoDestroyWindow() const;
```

Win	PM	Motif
Y	Y	Y

setAutoDeleteObject

Determines whether to delete the IWindow object when the presentation window is destroyed. The deletion occurs when the window system dispatches a destroy event to the window.

```
IWindow&
  setAutoDeleteObject( Boolean autoDelete = true );
```

Win	PM	Motif
Y	Y	Y

setAutoDestroyWindow

Determines whether to destroy the presentation window when the IWindow object is deleted. This is used in cases where an IWindow is a wrapper for a presentation window whose creation and destruction are handled outside the C++ object.

```
IWindow&
  setAutoDestroyWindow( Boolean autoDestroy = false );
```

Win	PM	Motif
Y	Y	Y

Observer Notification

Use these members to implement the INotifier protocol for IWindow classes.

notifyObservers

Notifies all observers in a part's collection.

```
virtual IWindow&
  notifyObservers( const INotificationEvent& event );
```

Win	PM	Motif
Y	Y	Y

Parent and Owner Support

Use these members to query and set a window's relationship and ownership.

handleWithParent

Returns the window handle for the specified window identifier and parent.

Note: This function replaces IWindow::handleWithId.

```
static IWindowHandle
  handleWithParent( unsigned long identifier,
                    const IWindowHandle& parent );
```

Win	PM	Motif
Y	Y	Y

Motif Motif's equivalent of window IDs are widget names. The User Interface Class Library has established the convention of numeric IDs (which the library converts to a character string internally) for portability.

Exceptions	
IInvalidParameter	The specified *parent* is invalid. You must specify a parent window. To specify the desktop window, use the handle of the window returned by IWindow::desktopWindow (p. 1049).

owner

Returns the window's owner. This function may return 0 if either of the following occurs:

- The window was created without an owner
- The owner window is not represented by an IWindow object

Because a pointer value of 0 can cause unpredictable behavior, check the value of the returned pointer before using it.

```
virtual IWindow*
  owner() const;
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, the User Interface Class Library keeps track of the owner in an IWindow class data member. Motif does not have owners, only parents. For portability, the User Interface Class Library has modeled the behavior to simulate owner handling in window systems that do not themselves support it.

parent

Returns the window's parent. This function may return 0 if the following occurs:

- The window was created without a parent
- The parent window is not represented by an IWindow object

Because a pointer value of 0 can cause unpredictable behavior, check the value of the returned pointer before using it.

```
IWindow*
  parent() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Motif This function ensures that the User Interface Class Library returns the correct parent, even for IWindows consisting of more than one widget combined (for example, IFrameWindow). The library continues up the widget chain until it finds a parent widget that is one of the following:

- An IWindow
- The desktopWindow
- The objectWindow

Use this function instead of XtParent wherever possible.

setOwner

Changes the window's owner.

```
virtual IWindow&
  setOwner( const IWindow* newOwner );
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, the User Interface Class Library keeps track of the owner in an IWindow class data member. Motif does not have owners, only parents. For portability, the User Interface Class Library has modeled the behavior to simulate owner handling in window systems that do not themselves support it.

setParent

Changes the window's parent.

```
virtual IWindow&
  setParent( const IWindow* newParent );
```

Win	PM	Motif
Y	Y	I

Motif Motif does not support this function. The X Window System does not support any way to re-parent widgets. Re-parenting X windows separately from their widgets, although possible, is considered too dangerous for general use. If you are trying to re-parent a window to effectively hide it, you must use IWindow::hide if you want system-independent, portable code.

Exceptions	
IInvalidParameter	The specified *newParent* is invalid. You must specify a nonzero parent window. To specify the desktop window, use the window returned by IWindow::desktopWindow (p. 1049).

windowWithOwner

Returns the IWindow object with the specified window identifier and owner. If *allThreads* is set to true, all threads are searched. Otherwise, only the current thread is searched. This is a static function.

Note: This function replaces IWindow::windowWithId.

If the specified window identifier is not associated with any IWindow object, 0 is returned.

```
static IWindow*
  windowWithOwner( unsigned long identifier,
                   const IWindow* owner,
                   Boolean allThreads = true );
```

Win	PM	Motif
Y	Y	Y

windowWithParent

Returns the IWindow object for the specified window identifier and parent. If *allThreads* is set to true, all threads are searched. Otherwise, only the current thread is searched. If you are searching for a primary or secondary window, you should specify IWindow::desktopWindow (p. 1049) for *parent*. This is a static function.

If the specified window identifier is not associated with any IWindow object, 0 is returned.

```
static IWindow*
  windowWithParent( unsigned long identifier,
                    const IWindow* parent,
                    Boolean allThreads = true );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The specified *parent* is invalid. You must specify a nonzero parent window. To specify the desktop window, use the window returned by IWindow::desktopWindow (p. 1049).

Pointer Capture Support

Use these members to query, capture, and release the pointer. If a window has captured the pointer, pointer events are only sent to the window that has captured the pointer even if the pointer is outside this window.

capturePointer

If *capture* is true, pointer events are sent only to this window even if the pointer is outside of this window. If *capture* is false, the window releases the pointer capture. If you attempt to capture the pointer when another window currently is capturing the pointer, an exception is thrown.

```
virtual IWindow&
  capturePointer( Boolean capture = true );
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The operating system is unable to capture the pointing device.

© IBM Corp. 1992, 1996

hasPointerCaptured

If this window is currently capturing pointer events, true is returned. Otherwise, false is returned.

```
virtual Boolean
  hasPointerCaptured() const;
```

Win	PM	Motif
Y	Y	N

releasePointer

Causes the window to release the pointer capture (pointer capture is set with the function capturePointer) (p. 1071).

This function causes mouse events to again be dispatched to the window underneath the mouse pointer.

```
virtual IWindow&
  releasePointer();
```

Win	PM	Motif
Y	Y	N

Pointer Positioning

Use these members to set and query the pointer's position.

movePointerTo

Sets the pointer to the point specified.

```
static void
  movePointerTo( const IPoint& position );
```

Win	PM	Motif
Y	Y	N

pointerPosition

Returns the position of the pointer.

```
static IPoint
  pointerPosition();
```

Win	PM	Motif
Y	Y	N

Sibling Order

Use these members to affect the ordering of windows with their siblings. The window order (or Z-order) is used when tabbing or painting sibling windows. Window painting starts from the bottom sibling and proceeds to the top window. Cursor tabbing occurs from the top sibling and proceeds to the bottom window.

defaultOrdering

Returns the order in which the library creates the new windows relative to their sibling windows.

```
static SiblingOrder
  defaultOrdering();
```

Win	PM	Motif
Y	Y	Y

positionBehindSibling

Puts this window in the Z-order behind the specified sibling window.

```
virtual IWindow&
  positionBehindSibling( const IWindowHandle& siblingWindow );
```

Win	PM	Motif
Y	Y	Y

positionBehindSiblings

Puts this window in the Z-order behind all its sibling windows.

```
virtual IWindow&
  positionBehindSiblings();
```

Win	PM	Motif
Y	Y	Y

positionOnSiblings

Puts this window in the Z-order on top of all its sibling windows.

```
virtual IWindow&
  positionOnSiblings();
```

Win	PM	Motif
Y	Y	Y

setDefaultOrdering

Determines whether new windows are created on top of their sibling windows or behind them. By default, the library creates windows behind their siblings. If you call this function using the enumerator IWindow::onTopOfSiblings (p. 1099), the library creates new windows on top of their siblings.

```
static void
  setDefaultOrdering( SiblingOrder order );
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the IWindow (p. 1044) class. Use these members to set and query window styles.

© IBM Corp. 1992, 1996

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, are returned if you set *extendedOnly* to true.

```
virtual unsigned long
  convertToGUIStyle( const IBitFlag& style,
                     Boolean extendedOnly = false ) const;
```

Win	PM	Motif
Y	Y	I

Window Enablement

Use these members to enable and disable a window and to determine the enablement of a window.

disable

Prevents keyboard and mouse input from being sent to the window.

```
virtual IWindow&
  disable();
```

Win	PM	Motif
Y	Y	Y

enable

Enables the window to accept keyboard and mouse input.

```
virtual IWindow&
  enable( Boolean enableWindow = true );
```

Win	PM	Motif
Y	Y	Y

isEnabled

If the window is sent mouse and keyboard input, true is returned. Otherwise, false is returned.

```
Boolean
  isEnabled() const;
```

Win	PM	Motif
Y	Y	Y

Window Painting

Use these members to display a window and determine the visibility of a window.

disableUpdate

Prevents changes to a window from being drawn on the screen. Typically, you can use this function to make a series of changes to a window without displaying each change as it is made. Call IWindow::show (p. 1077) to allow changes to draw on the screen. Do not destroy a window while it is in the update-disabled state. If you do, the library will not remove the destroyed window from the screen.

```
virtual IWindow&
  disableUpdate();
```

Win	PM	Motif
Y	Y	Y

Motif Motif ignores this function, because the X Window System is designed to avoid the need to disable or enable during most complex drawing.

enableUpdate

Enables or disables drawing changes to a window on the screen.

```
virtual IWindow&
  enableUpdate( Boolean enableWindow = true );
```

Win	PM	Motif
Y	Y	Y

Motif Motif ignores this function, because the X Window System is designed to avoid the need to disable or enable during most complex drawing.

hide

Hides the window.

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

Win	PM	Motif
Y	Y	Y

hideSourceEmphasis

Called by a pop-up menu handler to notify the window to remove pop-up menu emphasis. The library provides this function so that derived classes needing to remove pop-up menu emphasis can override it. The default behavior of IWindow is to do nothing.

```
virtual IWindow&
  hideSourceEmphasis();
```

Win	PM	Motif
Y	Y	Y

isShowing

If any part of the window is showing, true is returned. Otherwise, false is returned.

Note: A window can be visible and yet not be showing if it is covered by another window.

```
Boolean
  isShowing() const;
```

Win	PM	Motif
Y	Y	Y

Win This function returns the same result as isVisible (p. 1075).

isVisible

If the window's style is set to visible, true is returned. Otherwise, false is returned.

Note: A window can have the style visible (p. 1097) and yet not be showing if it is covered by another window.

© IBM Corp. 1992, 1996

```
Boolean
  isVisible() const;
```

Win	PM	Motif
Y	Y	Y

presSpace Acquires and returns the presentation space handle (also known as graphics context, or GC) for the window.

```
virtual IPresSpaceHandle
  presSpace() const;
```

Win	PM	Motif
Y	Y	Y

Motif This function returns a writable graphics context (GC) for this widget, as returned by the XtCreateGC(thisWidget, 0, 0) subroutine. If there has been a prior call to this function (without an intervening call to releasePresSpace), an existing GC is returned. This allows various client functions to draw using a shared GC so that line styles, fonts, and so forth are uniform. If this is not the behavior you want, you can create your own GC using X-toolkit functions.

Exceptions	
IAccessError	The operating system is unable to allocate a presentation space for the window.

refresh Invalidates or updates all or part of the window.

1
```
virtual IWindow&
  refresh( const IRectangle& invalidRectangle,
           Boolean immediate = false );
```

Win	PM	Motif
Y	Y	Y

Invalidates and redraws a specified window rectangle. If you specify true for *immediate*, this function paints the window synchronously. This means all painting is done before this function returns to the caller. If immediate is false, all painting may not be done when the function returns to the caller.

Motif This function is ignored in Motif, as the X Window System is designed to avoid the need for an application to cue the window system when to redraw screen areas.

According to O'Reilly Vol.4, p.25, "...Xt automatically redraws correctly written widgets at the appropriate times so that your application doesn't have to worry about this." All Motif widgets are "correctly written," and anything else is a user-written widget error.

2
```
virtual IWindow&
  refresh( RefreshType type = paintAll );
```

Win	PM	Motif
Y	Y	Y

Invalidates or updates all or part of the window, depending on the state of the *type* parameter. If type is set to IWindow::paintAll, then the entire window is invalidated. Painting may or may not occur before the call returns. If type is set to

IWindow::immediate, then the current invalidated area of the window is painted. Painting is completed before the call returns. If type is set to IWindow::paintAllImmediate, the entire window is invalidated and then painted. Painting is completed before the call returns.

releasePresSpace

Releases the window's presentation space handle.

```
virtual void
  releasePresSpace(
   const IPresSpaceHandle& presentationSpaceHandle) const;
```

Win	PM	Motif
Y	Y	Y

Motif This function frees the graphics context (GC), previously obtained via presSpace, using XtFreeGC. The GC freed by this function is shared, so it might not be freed until other client functions have also called releasePresSpace. See presSpace for details. If this is not the behavior you want, you can create your own GC using X-toolkit functions and free it using XtFreeGC.

Exceptions	
IInvalidRequest.	The operating system is unable to deallocate the presentation space.

show

Makes the window visible.

Note: Another window occurring earlier in the Z-order might still obscure this window.

```
virtual IWindow&
  show( Boolean showWindow = true );
```

Win	PM	Motif
Y	Y	Y

showSourceEmphasis

Called by a pop-up menu handler to notify the window to draw pop-up menu emphasis. The library provides this function so that derived classes needing to remove pop-up menu emphasis can override it. The default behavior of IWindow is to do nothing.

```
virtual IWindow&
  showSourceEmphasis( Boolean show = true );
```

Win	PM	Motif
Y	Y	Y

Window Positioning

Use these members to set and query the size and position of windows. Unless otherwise noted, the orientation of the coordinates accepted and returned by these members is the application orientation. For more information about coordinate orientation, see ICoordinateSystem (p. 230).

© IBM Corp. 1992, 1996

IWindow

mapPoint

Maps a point from one window's coordinate space to another's, such as mapping from screen coordinates to window coordinates.

Note: This is a static function.

```
static IPoint
  mapPoint( const IPoint& point,
            const IWindowHandle& from,
            const IWindowHandle& to );
```

Win	PM	Motif
Y	Y	Y

moveSizeTo

Changes the position and size of the window.

Note: If you override moveSizeTo, moveTo, or sizeTo, you should call the inherited version of the function you override during your processing. Since these functions are implemented in terms of each other in some environments, you could cause recursion if, for example, you implement part of your version of moveSizeTo by calling IWindow::moveTo and IWindow::sizeTo. Instead, perform your processing and then call IWindow::moveSizeTo.

```
virtual IWindow&
  moveSizeTo( const IRectangle& newSizeAndPosition );
```

Win	PM	Motif
Y	Y	Y

moveTo

Changes the position of the window.

Note: If you override moveSizeTo, moveTo, or sizeTo, you should call the inherited version of the function you override during your processing. Since these functions are implemented in terms of each other in some environments, you could cause recursion if, for example, you implement part of your version of moveSizeTo by calling IWindow::moveTo and IWindow::sizeTo. Instead, perform your processing and then call IWindow::moveSizeTo.

```
virtual IWindow&
  moveTo( const IPoint& newPosition );
```

Win	PM	Motif
Y	Y	Y

nativeRect

Returns a rectangle representing the position and size of the window. Unlike IWindow::rect (p. 1079), this function always returns the position in the native GUI orientation.

Note: This function returns IRectangle(0,0,0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual IRectangle
  nativeRect() const;
```

Win	PM	Motif
Y	Y	N

parentSize Returns an ISize object representing the size of the client rectangle in the parent window. The *client rectangle* is the coordinate space used by child windows to specify their location. If the current window or parent is determined to be the desktop, the size of the desktop is returned.

1
```
virtual ISize
  parentSize() const;
```

Win	PM	Motif
Y	Y	N

Returns the size of the parent of this window.

2
```
static ISize
  parentSize( const IWindowHandle& windowHandle );
```

Win	PM	Motif
Y	Y	N

Returns the size of the parent window of the window with the handle *windowHandle*.

position Returns the position of the window.

```
virtual IPoint
  position() const;
```

Win	PM	Motif
Y	Y	Y

rect Returns a rectangle representing the position and size of the window. If the application uses a different coordinate orientation than the window system orientation, this function converts from native window system coordinates to the application orientation. Refer to ICoordinateSystem (p. 230) for information about coordinate orientation.

Note: This function returns IRectangle(0,0,0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual IRectangle
  rect() const;
```

Win	PM	Motif
Y	Y	Y

size Returns the size of the window in window coordinates.

Note: This function returns ISize(0,0) for a frame window if it is constructed using the shell position and the window has not been shown.

```
virtual ISize
  size() const;
```

Win	PM	Motif
Y	Y	Y

sizeTo Changes the size of the window.

Note: If you override moveSizeTo, moveTo, or sizeTo, you should call the inherited version of the function you override during your processing. Since these functions are implemented in terms of each other in some environments, you

© IBM Corp. 1992, 1996

could cause recursion if, for example, you implement part of your version of moveSizeTo by calling IWindow::moveTo and IWindow::sizeTo. Instead, perform your processing and then call IWindow::moveSizeTo.

```
virtual IWindow&
  sizeTo( const ISize& newSize );
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Compound Control

Use these members to access the top window system. A *compound control* is a window that consists of more than one window system control.

isRelatedHandle

If the specified handle is one of those that this IWindow derived class created in its constructor, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isRelatedHandle( const IWindowHandle& windowHandle ) const;
```

Win	PM	Motif
N	N	Y

Constructors

You can construct and destruct objects of the IWindow class. You cannot copy or assign IWindow objects because both the copy constructor and the assignment operator are private functions.

IWindow

```
IWindow();
```

Win	PM	Motif
Y	Y	Y

Used by derived classes to create objects of this class.

Drag and Drop Support

Use these members to query and set the drag-item provider for objects of this class.

isDragStarting

The *event* is either a mouse button down, mouse button up, or a mouse move message. This member function exists to help determine whether or not a drag condition is occurring for this control. If proper drag conditions are met then true is returned. Otherwise, false is returned.

The default implementation looks for a button down event. If the mouse then moves more than two pels before a mouse button event is received, then a drag is occurring. If the criteria for a drag from a derived control is more elaborate than the default implementation, then the derived class needs to override this member function.

```
virtual Boolean
  isDragStarting( IEvent& event );
```

Win	PM	Motif
Y	N	N

Event-Handling Implementation

Event-handling implementation members perform processing needed to allow handlers to properly receive GUI events and to route these events.

addRelatedHandleToWindowSet

Informs the User Interface Class Library that the specified window handle is part of this IWindow object, as opposed to part of another object or a "raw" window not wrapped by any IWindow object. The User Interface Class Library uses this function internally. If you write classes derived from IWindow for new controls, you might also need to use this function internally. Do so only when the IWindow object is built from more than one window system control.

```
static void
  addRelatedHandleToWindowSet(
   IWindow* window,
   const IWindowHandle& windowHandle );
```

Win	PM	Motif
N	N	Y

© IBM Corp. 1992, 1996

IWindow

addToWindowSet

Adds a window to the current thread's collection of windows. You must provide a valid window handle.

```
static void
  addToWindowSet( IWindow* window,
                  const IWindowHandle& windowHandle );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	Either the specified *window* or *windowHandle* is invalid.

defaultProcedure

Provides default processing behavior for a window event. This function calls the window procedure that was returned from the window derived class during the call to IWindow::startHandlingEventsFor (p. 1084).

```
IWindow&
  defaultProcedure( IEvent& event );
```

Win	PM	Motif
Y	Y	Y

Motif In Motif, this function is not called during normal event processing and has no effect if it is called. There is no default window procedure that you must explicitly call or that you can avoid by not calling it. The window itself (that is, the widget) determines what default or standard processing occurs prior to application code (such as the User Interface Class Library) being called. The window also determines what processing occurs afterward. Additionally, you cannot prevent X-Motif from calling other portions of application code, which might be registered with this widget as callbacks. You cannot affect the order reliably. However, widgets and other callbacks are designed with this fact in mind.

deleteIsInProcess

If the IWindow object destructor has been called or the presentation window is being destroyed, true is returned. Otherwise, false is returned.

```
Boolean
  deleteIsInProcess() const;
```

Win	PM	Motif
Y	Y	Y

dispatch

Dispatches events to the handlers associated with a window. If a handler does not process the event or no handlers exist for the window, the library calls IWindow::defaultProcedure (p. 1082) to process the event.

The library also performs post-processing for certain events. For example, the library uses destroy events to clean up the related IWindow object, if needed. The library uses resize events to verify that re-coordination is current.

```
Boolean
  dispatch( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

Motif Dispatches the event in the following order:

1. To the handlers in the list
2. To the owner, because Motif does not have the concept of owners
3. To Motif, so any other callbacks for this widget might be called

In Motif, there is no default window procedure that must be explicitly called or that can be avoided by not calling it. The window itself (the widget) determines what default or standard processing occurs prior to application code (such as the User Interface Class Library) being called. The window also determines what processing occurs afterward. It is not possible to prevent other bits of application code, which might be registered with this widget as callbacks, from being called by X-Motif. You cannot affect the order reliably. However, widgets and other callbacks are designed with this fact in mind.

Exceptions	
IInvalidRequest	The window dispatcher found a handler that was invalid. The handler must be removed prior to deletion.

passEventToOwner

Determines if an event is passed on to the owner.

```
virtual Boolean
  passEventToOwner( IEvent& event );
```

Win	PM	Motif
N	*N*	*Y*

registerCallbacks

Adds callbacks and X event handlers to this IWindow for events it receives. This function adds all possible callbacks and X event handlers. IHandler derived classes later determine which events they will process. You *must* call Inherited::registerCallbacks so the handler objects can add their processing.

The implementation in IWindow itself adds callbacks and X event handlers for events every IWindow can receive.

```
virtual void
  registerCallbacks();
```

Win	PM	Motif
N	*N*	*Y*

© IBM Corp. 1992, 1996

IWindow

removeFromWindowSet

Removes a window from the current thread's collection of windows.

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

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The specified *window* is invalid.

removeRelatedHandleFromWindowSet

Informs the User Interface Class Library that the specified window handle is no longer part of this IWindow object. The library uses this function internally. Users writing classes derived from IWindow for new controls might also need to use this function internally. It should only be used when the IWindow object is built from more than one window system control.

```
static void
  removeRelatedHandleFromWindowSet(
   IWindow* window,
   const IWindowHandle& windowHandle );
```

Win	PM	Motif
N	N	Y

startHandlingEventsFor

Starts processing events for the window. This is a combination of subclassing the window procedure and storing the window in the current thread's collection of windows. You can only call this function once per window, typically during construction of the window.

1

```
IWindow&
  startHandlingEventsFor( const IWindowHandle& windowHandle );
```

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the logic is somewhat different from other graphical user interfaces, such as OS/2 Presentation Manager. Motif does not use a single connection point like the Presentation Manager window procedure. It uses individual callbacks, one or more per window or control, which equate roughly to the types of events that windows or controls might receive.

Exceptions	
IInvalidParameter	The specified *windowHandle* is invalid, or identifies a window for which a window object already exists.
IInvalidRequest	The default procedure for the window could not be replaced. Possibly a window object already exists for the window identified by *windowHandle*, or the window was created by a different process or application.

2 `IWindow&`
`  startHandlingEventsFor( unsigned long identifier,`
`                          IWindow* parent );`

Win	PM	Motif
Y	Y	Y

Motif In X-Motif, the logic is somewhat different from other graphical user interfaces, such as OS/2 Presentation Manager. Motif does not use a single connection point like the Presentation Manager window procedure. It uses individual callbacks, one or more per window or control, which equate roughly to the types of events that windows or controls might receive.

Exceptions	
IInvalidParameter	The specified *parent* is invalid. You must specify a nonzero parent window.
IInvalidRequest	No window with the specified *identifier* could be found.

unregisterCallbacks

Removes callbacks and X event handlers from this IWindow, which were added by registerCallbacks (p. 1083). You *must* call Inherited::unregisterCallbacks so the handler objects can remove their processing.

```
virtual void
  unregisterCallbacks();
```

Win	PM	Motif
N	N	Y

Implementation

These members provide utilities used to implement this class.

addHandler

Adds an IHandler object to the window.

```
IWindow&
  addHandler( IHandler* newHandler );
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IInvalidParameter	The specified handler, *newHandler*, is invalid.

color

Returns the color value of the specified window area. If the window area has no color set, the area's color is set to the specified default color and the default color is returned. You identify the window area using a system-defined presentation parameter value.

The arguments are the following:

© IBM Corp. 1992, 1996

colorArea
The area for which the color is being queried. The values are OS/2 Presentation Manager Toolkit presentation parameters color areas, such as PP_FOREGROUNDCOLOR. It is not limited to these values; you can define your own, but then you must also supply the support for these new values. You can only choose the values referring to RGB colors; you cannot use ones referring to index colors.

defaultColor
If you specify *defaultColor* and the window area has no color set, this color is returned.

Note: This function is obsolete and may be removed in a future release of the library. The User Interface Class Library provides this function for backward compatibility only. Instead, you should use the explicit color setting, resetting, and querying functions, such as setBackgroundColor.

1
```
IColor
  color( unsigned long colorArea,
         const IColor& defaultColor ) const;
```

Win	PM	Motif
Y	*Y*	*N*

2
```
IColor
  color( unsigned long colorArea ) const;
```

Win	PM	Motif
Y	*Y*	*N*

Exceptions	
IInvalidRequest	A color could not be found for the specified color area.

create

Calls the graphical presentation system to create a window object.

1
```
virtual IWindowHandle
  create(
   unsigned long id,
   const char* text,
   unsigned long style,
   IXmCreateFunction createFunction,
   const IWindowHandle& parent,
   const IWindowHandle& owner,
   const IRectangle& initRect,
   const void* callerArgList,
   const unsigned int callerNumberArguments,
   IWindow::SiblingOrder ordering = defaultOrdering ( ) );
```

Win	PM	Motif
N	*N*	*Y*

Generally, classes derived from IWindow that were implemented for Motif do not use this function. Instead, these classes use XtCreateWidget or an XmCreate... helper routine.

2

```
virtual IWindowHandle
  create(
   unsigned long id,
   const char* text,
   unsigned long style,
   const char* windowClass,
   const IWindowHandle& parent,
   const IWindowHandle& owner,
   const IRectangle& initRect,
   const void* ctlData,
   const void* presParams,
   IWindow::SiblingOrder ordering = defaultOrdering ( ),
   unsigned long extendedStyle = 0 );
```

Win	PM	Motif
Y	Y	N

PM You should limit *id* to values between 0 and 65535. Using other values can give unpredictable results.

Exceptions	
IAccessError	The operating system is unable to create the window.

extendedStyle

Returns an unsigned long representing the window's extended style.

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

Win	PM	Motif
Y	Y	I

isPrimaryWindow

If this window has no owner and the parent is the desktop (root) window, true is returned. Otherwise, false is returned.

```
Boolean
  isPrimaryWindow() const;
```

Win	PM	Motif
Y	Y	Y

prepareForUse

Prepares this object to use the specified window object (created elsewhere). Typically, this function is called from a wrapper constructor (for example, one that takes only a window handle).

```
virtual IWindow&
  prepareForUse( const IWindowHandle& windowHandle );
```

Win	PM	Motif
N	N	Y

© IBM Corp. 1992, 1996

removeHandler

Removes a previously added handler.

```
IWindow&
  removeHandler( IHandler* oldHandler );
```

Win	PM	Motif
Y	Y	Y

resetColor

Resets the color of a window back to its default value by undoing any previous calls to setColor.

```
IWindow&
  resetColor( unsigned long colorArea );
```

Win	PM	Motif
Y	Y	N

Exceptions	
IAccessError	The operating system is unable to reset the color.

setColor

Sets the window area to the specified color. You can identify the window area using a system-defined presentation parameter value.

The arguments are the following:

colorArea
: The area for which the color is being set. The values are OS/2 Presentation Manager Toolkit presentation parameters color areas, such as PP_FOREGROUNDCOLOR. It is not limited to these values; you can define your own, but then you must also supply the support for these new values. You can only choose the values that refer to RGB colors; you cannot use ones that refer to index colors.

color
: The area to be set to this color.

Note: This function is obsolete and may be removed in a future release of the library. The User Interface Class Library provides this function for backward compatibility only. Instead, you should use the explicit color setting, resetting, and querying functions, such as setBackgroundColor.

```
IWindow&
  setColor( unsigned long colorArea,
            const IColor& color );
```

Win	PM	Motif
Y	Y	N

setExtendedStyle

Sets the window's extended style.

```
virtual IWindow&
  setExtendedStyle( unsigned long extendedStyle );
```

Win	PM	Motif
Y	Y	I

setStyle

Sets the window's style.

```
virtual IWindow&
  setStyle( unsigned long style );
```

Win	PM	Motif
Y	Y	I

Motif This function is ignored in AIX. The User Interface Class Library recommends using individual functions to set the various states of an IWindow::Style (p. 1105) object, rather than the setStyle function. Use of this function will not result in portable code.

setWindowData

Sets the specified window word as an unsigned short or unsigned long. The result indicates whether it was successful.

1
```
IWindow&
  setWindowData( long index,
                 unsigned short ushort );
```

Win	PM	Motif
Y	Y	N

Motif This function is not supported in AIX. Motif widgets do not all have a window word; widgets of type Primitive have a word of user data (XmNuserData), but not all IWindows are based on Primitive widgets. Also, the C++ technique for adding more data to an object is to derive from that class and add your own private data members. This is the technique recommended by the User Interface Class Library to achieve portable code.

Exceptions	
IAccessError	The operating system is unable to set the unsigned short in the window data.

2
```
IWindow&
  setWindowData( long index,
                 unsigned long ulong );
```

Win	PM	Motif
Y	Y	N

Motif This function is not supported in AIX. Motif widgets do not all have a window word; widgets of type Primitive have a word of user data (XmNuserData), but not all IWindows are based on Primitive widgets. Also, the C++ technique for adding more data to an object is to derive from that class and add your own private data members. This is the technique recommended by the User Interface Class Library to achieve portable code.

© IBM Corp. 1992, 1996

Exceptions	
IAccessError	The operating system is unable to set the unsigned long in the window data.

style Returns an unsigned long representing the window's style.

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

Win	PM	Motif
Y	*Y*	*I*

Motif This function is ignored; it returns 0 always. The User Interface Class Library recommends using individual functions to query the various states of an IWindow::Style (p. 1105) object, rather than the style function. Use of this function will not result in portable code.

windowULong

Returns the specified window word as an unsigned long.

```
unsigned long
  windowULong( long index ) const;
```

Win	PM	Motif
Y	*Y*	*N*

Motif This function is not supported in AIX. Motif widgets do not all have a window word; widgets of type Primitive have a word of user data (XmNuserData), but not all IWindows are based on Primitive widgets. Also, the C++ technique for adding more data to an object is to derive from that class and add your own private data members. This is the technique recommended by the User Interface Class Library to achieve portable code.

windowUShort

Returns the specified window word as an unsigned short.

```
unsigned short
  windowUShort( long index ) const;
```

Win	PM	Motif
Y	*Y*	*N*

Motif This function is not supported in AIX. Motif widgets do not all have a window word; widgets of type Primitive have a word of user data (XmNuserData), but not all IWindows are based on Primitive widgets. Also, the C++ technique for adding more data to an object is to derive from that class and add your own private data members. This is the technique recommended by the User Interface Class Library to achieve portable code.

Layout Support

Layout support members supply information used by the canvas classes to provide dialog-like behavior. The virtual functions contain minimal implementations, which can be overridden in derived classes.

calcMinimumSize

Returns the recommended minimum size of this IWindow object. The library uses this size in the canvas layout routine. If no derived classes override this function, the default size is 100 x 100 pels.

```
virtual ISize
  calcMinimumSize() const;
```

Win	PM	Motif
Y	Y	Y

savedMinimumSize

Returns the calculated minimum size of this IWindow object saved using IWindow::saveMinimumSize (p. 1091). As an optimization for not calling calcMinimumSize (p. 1091), IWindow::mininumSize (p. 1065) may return the value returned by this function. See IWindow::minimumSize and enableMinimumSizeCaching (p. 1064) for details.

```
ISize
  savedMinimumSize() const;
```

Win	PM	Motif
Y	Y	N

saveMinimumSize

Saves the calculated minimum size of this IWindow object. IWindow::minimumSize (p. 1065) uses this function to store values returned by calcMinimumSize (p. 1091). See IWindow::minimumSize for details.

```
IWindow&
  saveMinimumSize( const ISize& size );
```

Win	PM	Motif
Y	Y	N

Observer Notification

Use these members to implement the INotifier protocol for IWindow classes.

addObserver

Adds an observer to the part's collection.

```
virtual IWindow&
  addObserver( IObserver& observer,
               const IEventData& userData = IEventData ( 0 ) );
```

Win	PM	Motif
Y	Y	Y

notificationHandler

Returns this window's notification handler. If the window does not have a notification handler, 0 is returned. IWindow creates the notification handler when processing the first enableNotification (p. 1067) request.

© IBM Corp. 1992, 1996

IWindow

```
IWindowNotifyHandler*
  notificationHandler() const;
```

Win	PM	Motif
Y	Y	Y

Notifies all observers in a part's collection.

notifyObservers

```
virtual IWindow&
  notifyObservers( const INotificationId& notification );
```

Win	PM	Motif
Y	Y	Y

observerList

Returns the collection of IObservers.

```
IObserverList&
  observerList() const;
```

Win	PM	Motif
Y	Y	Y

removeAllObservers

Removes all observers from the part's collection.

```
virtual IWindow&
  removeAllObservers();
```

Win	PM	Motif
Y	Y	Y

removeObserver

Removes an observer from the part's collection.

```
virtual IWindow&
  removeObserver( IObserver& observer );
```

Win	PM	Motif
Y	Y	Y

setNotificationHandler

Stores the window's notification handler. This handler captures window events and notifies observers of the window.

```
IWindow&
  setNotificationHandler(
   IWindowNotifyHandler* notifyHandler);
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Public Data

Notification Members

Use these members to identify and enable notifications sent to observer objects.

activeColorId Notification identifier provided to observers when the active color of a window changes.

```
static INotificationId const
  activeColorId;
```

Win	PM	Motif
I	*Y*	*Y*

backgroundColorId

Notification identifier provided to observers when the background color of a window changes.

```
static INotificationId const
  backgroundColorId;
```

Win	PM	Motif
Y	*Y*	*Y*

borderColorId

Notification identifier provided to observers when the border color of a window changes.

```
static INotificationId const
  borderColorId;
```

Win	PM	Motif
I	*Y*	*Y*

commandId Notification identifier provided to observers when an IWindow::command event occurs. IWindow provides a pointer to the IEvent in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  commandId;
```

Win	PM	Motif
Y	*Y*	*Y*

deleteId Notification identifier provided to observers when the window object is deleted.

Note: IWindow sends this notification from its destructor. This means that the derived portions of the window have already been deleted. You should, therefore, not cast the pointer to the notifier data to an object that is derived from IWindow. In addition, the underlying system window may not be available at this time. Do not call any function that requires the system window unless you verify the system window exists. See IWindow::isValid (p. 1050).

© IBM Corp. 1992, 1996

```
static INotificationId const
  deleteId;
```

Win	PM	Motif
Y	Y	Y

disabledBackgroundColorId

Notification identifier provided to observers when the disabled background color of a window changes.

```
static INotificationId const
  disabledBackgroundColorId;
```

Win	PM	Motif
I	Y	Y

disabledForegroundColorId

Notification identifier provided to observers when the disabled foreground color of a window changes.

```
static INotificationId const
  disabledForegroundColorId;
```

Win	PM	Motif
I	Y	Y

enableId

Notification identifier provided to observers when the keyboard- and mouse-enabled state of the window changes. IWindow provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value will be true if the window is now enabled and false if the window is now disabled.

```
static INotificationId const
  enableId;
```

Win	PM	Motif
Y	Y	Y

focusId

Notification identifier provided to observers when the window receives or loses the focus. IWindow provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the window now has the focus and false if the window just lost the focus.

```
static INotificationId const
  focusId;
```

Win	PM	Motif
Y	Y	Y

fontId

Notification identifier provided to observers when the font of a window changes.

```
static INotificationId const
  fontId;
```

Win	PM	Motif
Y	Y	Y

foregroundColorId

Notification identifier provided to observers when the foreground color of a window changes.

```
static INotificationId const
  foregroundColorId;
```

Win	PM	Motif
Y	*Y*	*Y*

hiliteBackgroundColorId

Notification identifier provided to observers when the highlight background color of a window changes.

```
static INotificationId const
  hiliteBackgroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

hiliteForegroundColorId

Notification identifier provided to observers when the highlight foreground color of a window changes.

```
static INotificationId const
  hiliteForegroundColorId;
```

Win	PM	Motif
I	*Y*	*Y*

inactiveColorId

Notification identifier provided to observers when the inactive color of a window changes.

```
static INotificationId const
  inactiveColorId;
```

Win	PM	Motif
I	*Y*	*Y*

positionId

Notification identifier provided to observers when the position of a window changes. When both the size and position of a window changes, IWindow sends the size change notification first, followed by the position change notification. IWindow provides a pointer to an IPoint object for the new position in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  positionId;
```

Win	PM	Motif
Y	*Y*	*Y*

shadowColorId

Notification identifier provided to observers when the shadow color of a window changes.

```
static INotificationId const
  shadowColorId;
```

Win	PM	Motif
I	*Y*	*Y*

© IBM Corp. 1992, 1996

sizeId Notification identifier provided to observers when the size of a window changes. When both the size and position of a window changes, IWindow sends the size change notification first, followed by the position change notification. IWindow provides a pointer to an ISize object for the new size in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  sizeId;
```

Win	PM	Motif
Y	Y	Y

systemCommandId

Notification identifier provided to observers when an IWindow::systemCommand event occurs. IWindow provides a pointer to the IEvent in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

```
static INotificationId const
  systemCommandId;
```

Win	PM	Motif
Y	Y	Y

visibleId Notification identifier provided to observers when the visibility of a window changes. IWindow provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the window is now visible and false if it is now hidden.

```
static INotificationId const
  visibleId;
```

Win	PM	Motif
Y	Y	Y

Styles

These style members provide a set of valid styles for the IWindow (p. 1044) class. Use these members to set and query window styles.

clipChildren Excludes the area occupied by the children of the window when drawing in the window. Child windows are always clipped to their parent window. When the system paints a parent window, this style controls whether the invalidated region of the parent window includes the area occupied by its children, thus preventing a window from painting over its child windows.

Note: Do not use this style for the parent of an IComboBox that already has the style simpleType (p. 72).

```
static const Style
  clipChildren;
```

Win	PM	Motif
Y	Y	Y

clipSiblings When the library displays multiple siblings, this style controls which sibling window the library displays on top. Sibling windows are windows that share the same parent window. Assign this style to the sibling window that the library should display on top of the other siblings, in Z-order.

```
static const Style
  clipSiblings;
```

Win	PM	Motif
Y	*Y*	*Y*

clipToParent Allows a window to paint outside of its window boundary up to the window boundary of its parent.

```
static const Style
  clipToParent;
```

Win	PM	Motif
I	*Y*	*Y*

disabled Specifies that keyboard and mouse input are no longer dispatched to the window, preventing the window from being used.

```
static const Style
  disabled;
```

Win	PM	Motif
Y	*Y*	*Y*

noStyle Sets all styles off.

```
static const Style
  noStyle;
```

Win	PM	Motif
Y	*Y*	*Y*

saveBits Optimizes the painting of a window by saving the screen image of the area under the window as a bitmap and then using the bitmap to redraw the window when necessary.

```
static const Style
  saveBits;
```

Win	PM	Motif
I	*Y*	*Y*

synchPaint Synchronously repaints the window. Without this style, painting only occurs if there are no other events waiting to be processed.

Note: The only way to turn off this style is to call the protected member function IWindow::setStyle (p. 1089) to reset it after you create the window.

```
static const Style
  synchPaint;
```

Win	PM	Motif
I	*Y*	*Y*

visible Causes the window to be visible. In general, controls are constructed as visible and frame windows as invisible.

© IBM Corp. 1992, 1996

```
static const Style
  visible;
```

Win	PM	Motif
Y	Y	Y

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IWindow contains the following nested classes:

IWindow::Style (see page 1105)
IWindow::ChildCursor (see page 1100)
IWindow::ExceptionFn (see page 1103)

EventType

```
EventType {
    command = 1,   systemCommand, control,       help,          character
    };
```

Use these enumerators to specify windowing system message identifiers:

command
Specifies the user command event type.

systemCommand
Specifies the system command event type.

help
Specifies the help event type.

character
Specifies the key character event type.

Win The EventType values map to the following system defined messages:

command
WM_COMMAND. For processing by a command handler, parameter 2 must be 0. Non zero values for parameter 2 are defined by the operating system.

systemCommand
WM_SYSCOMMAND.

help
WM_HELP.

character
WM_CHAR.

Layout

```
Layout {
    windowCreated = 1,            colorChanged = 2,
    sizeChanged = 4,              minimumSizeChanged = 8,
    childMinimumSizeChanged = 16, fontChanged = 32,
    fontPropogated = 64,          layoutChanged = 128,
    immediateUpdate = 256,        childWindowCreated = 512,
    windowDestroyed = 1024,       childWindowDestroyed = 2048
    };
```

Use the preceding enumerators to specify window layout values.

RefreshType

```
RefreshType {
    paintAll,
    immediate,
    paintAllImmediate
    };
```

Use these enumerators to specify the refresh type when updating a window using refresh (p. 1076).

paintAll
Invalidates the entire window. Painting occurs as part of the normal screen update processing provided by the operating system.

immediate
Causes a synchronous paint of the current invalidated region of the window. This means painting is complete when refresh returns.

paintAllImmediate
Causes a synchronous paint of the entire window. This means painting is complete when refresh returns.

SiblingOrder

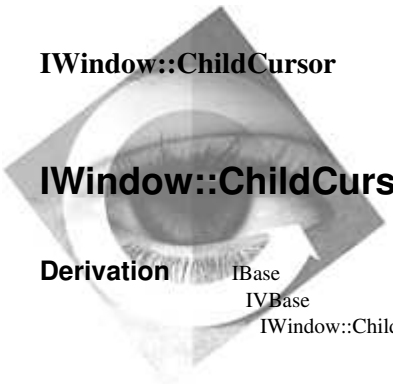
```
SiblingOrder {
    onTopOfSiblings,
    behindSiblings
    };
```

Use these enumerators to specify the sibling order of newly created windows:

onTopOfSiblings
Creates new windows on top of their sibling windows.

behindSiblings
Creates new windows behind their sibling windows.

© IBM Corp. 1992, 1996



IWindow::ChildCursor

Derivation

IBase
IVBase
IWindow::ChildCursor

Inherited By None.

Header File iwindow.hpp

Members

Member	Page	Member	Page
Constructor	1100	setToFirst	1101
ChildCursor	1100	setToNext	1101
invalidate	1101	~ChildCursor	1100
isValid	1101		

Use the nested class IWindow::ChildCursor to iterate over the children of any window. The library accesses the children in Z-order, from top to bottom.

Public Functions

Constructors

You can construct and destruct objects of the IWindow::ChildCursor class. You cannot copy or assign IWindow::ChildCursor objects because both the copy constructor and the assignment operator are private functions.

ChildCursor Constructs objects of the IWindow::ChildCursor class. You can use the resulting object to enumerate the child windows of the specified window by using this object with IWindow::childAt (p. 1052).

```
ChildCursor( IWindow& parent );
```

Win	PM	Motif
Y	Y	Y

You can construct objects of this class from a reference to the window over whose child windows you want to iterate.

~ChildCursor

```
virtual
  ~ChildCursor();
```

Win	PM	Motif
Y	Y	Y

© Copyright IBM Corp. 1992, 1996

Exceptions	
IAccessError	The operating system failed to deallocate resources used by the cursor.

Cursor Movement

Use these members to control cursor movement.

setToFirst Resets the cursor position to the first child window (in Z-order).

```
virtual Boolean
  setToFirst();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system failed to allocate resources for the cursor; possible invalid window handle.

setToNext Advances the cursor position to the next child window (in Z-order).

```
virtual Boolean
  setToNext();
```

Win	PM	Motif
Y	Y	Y

Cursor Validation

Use these members to query or set the validity of the cursor.

invalidate Marks the cursor as invalid.

```
virtual void
  invalidate();
```

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The operating system failed to deallocate resources used by the cursor.

isValid If the cursor is in a valid area, true is returned. Otherwise, false is returned.

```
virtual Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

© IBM Corp. 1992, 1996

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IWindow::ExceptionFn

Derivation

IBase
  IVBase
    IWindow::ExceptionFn

Inherited By None.

Header File iwindow.hpp

Members

Member	Page
handleException	1103
~ExceptionFn	1103

The nested class IWindow::ExceptionFn processes exceptions occurring when events are dispatched by the IWindow (p. 1044) dispatcher.

Public Functions

Constructors

Use this member to delete objects of the IWindow::ExceptionFn class.

~ExceptionFn

```
virtual
  ~ExceptionFn();
```

Win	PM	Motif
Y	Y	Y

Exception Processing

Use these members to process C++ exceptions in the IWindow dispatcher.

handleException

If an exception-function object is registered with IWindow::setExceptionFunction (p. 1062), this virtual function of the object is called when an uncaught exception occurs while dispatching events. If the exception situation was handled and it is safe to continue dispatching events, this function should return true. If this function returns false, the exception is thrown again.

© Copyright IBM Corp. 1992, 1996

IWindow::ExceptionFn

```
virtual Boolean
  handleException( IException& dispatcherException,
                   IEvent& exceptionEvent ) = 0;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IWindow::Style

Derivation

IBase
  IBitFlag
    IWindow::Style

Inherited By None.

Header File iwindow.hpp

The nested class IWindow::Style provides a set of valid styles for the IWindow (p. 1044) class.

Motif The visible style corresponds to the X-Motif state "mapped" and the disable style corresponds to "insensitive."

Motif ignores the following styles, which appear set to on:

- clipSiblings
- synchPaint

Motif ignores the following styles, which appear set to off:

- clipChildren
- clipToParent
- saveBits

The intent of the synchPaint style is to ensure painting is not delayed by other events on the queue. In Motif, control of painting at this level is completely the responsibility of the widget (control) itself. The programmer does not control it. Typically, a Motif control ensures timely painting via the X event filter for exposure (compress_exposure) set to XtExposeCompressMultiple. Therefore, the library ignores the setting of this style and always returns it set on.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	**messageFile**	**setMessageFile**

© Copyright IBM Corp. 1992, 1996

IWindow::Style

IBase		
asString	**messageText**	**version**

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

IWindowHandle

Derivation

IBase
  IHandle
    IWindowHandle

Inherited By None.

Header File ihandle.hpp

Members

Member	Page	Member	Page
Constructor	1107	postEvents	1108
isValid	1109	sendEvent	1109
operator _WidgetRec *	1108	sendEvents	1109
postEvent	1108		

The IWindowHandle class accesses windows.

PM IWindowHandle is an alias for the Presentation Manager Toolkit type HWND.

Motif IWindowHandle is an alias for a widget data type.

Public Functions

Constructors

You can construct objects of this class.

IWindowHandle

1 `IWindowHandle( Value hwnd = 0 );`

Win	PM	Motif
Y	*Y*	*Y*

You can construct objects of this class from a window handle (a value of type IHandle::Value), which defaults to 0.

2 `IWindowHandle( _WidgetRec* hwnd );`

Win	PM	Motif
N	*N*	*Y*

In Motif, you can construct objects of this class from an object of the X-Toolkit type Widget.

© Copyright IBM Corp. 1992, 1996

3 `IWindowHandle( int hwnd );`

Win	PM	Motif
N	N	Y

You can construct objects of this class from a window handle.

Operators

This group contains operators for this class.

operator _WidgetRec *

Returns the handle value as a native X-Toolkit window handle object.

```
operator _WidgetRec *() const;
```

Win	PM	Motif
N	N	Y

Sending and Posting Events

Use these members to send or post an event to a window handle or handles.

postEvent Posts an event constructed from the parameters to the window identified by this handle.

```
void
  postEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0ul,
             const IEventParameter2& parm2 = 0ul ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) posted using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when posting a user defined eventId (or message), the high word of the first parameter is lost.

postEvents Posts one event to multiple windows.

```
void
  postEvents( unsigned long eventId,
              const IEventParameter1& parm1 = 0ul,
              const IEventParameter2& parm2 = 0ul,
              BroadcastTo value = descendants ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) posted using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when posting a user defined eventId (or message), the high word of the first parameter is lost.

sendEvent Sends an event constructed from the parameters to the window identified by this handle.

```
IEventResult
  sendEvent( unsigned long eventId,
             const IEventParameter1& parm1 = 0ul,
             const IEventParameter2& parm2 = 0ul ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) sent using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when sending a user defined eventId (or message), the high word of the first parameter is lost.

sendEvents Broadcasts one event to multiple windows.

```
void
  sendEvents( unsigned long eventId,
              const IEventParameter1& parm1 = 0ul,
              const IEventParameter2& parm2 = 0ul,
              BroadcastTo value = descendants ) const;
```

Win	PM	Motif
Y	Y	Y

W32s Events (or messages) sent using this member function are subject to the restrictions and limitations of the Win32s system.

For example, when sending a user defined eventId (or message), the high word of the first parameter is lost.

Testing

Use these members to test a window handle.

isValid If the window handle is valid, true is returned.

```
Boolean
  isValid() const;
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

© IBM Corp. 1992, 1996

IWindowHandle

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	**unrecoverable**	

BroadcastTo

```
BroadcastTo {
    descendants,
    frames,
    frameDescendants
    };
```

Use these enumerators to specify post and send event-broadcasting attributes:

descendants
Broadcasts the event to all the descendants of this window handle.

frames
Broadcasts the event to frame windows only.

frameDescendants
Broadcasts the event to all the frame window descendants of this window handle.

IWindowNotifyHandler

Derivation

IBase
  IVBase
    IHandler
      IWindowNotifyHandler

Inherited By

IContainerControlNotifyHandler
IFrameWindowNotifyHandler
IListBoxNotifyHandler
IMenuNotifyHandler
IMMDeviceNotifyHandler
INotebookNotifyHandler
INumericSpinButtonNotifyHandler
IProgressIndicatorNotifyHandler
IScrollBarNotifyHandler
ITextControlNotifyHandler
ITextSpinButtonNotifyHandler

Header File iwinnhdr.hpp

Members

Member	Page
Constructor	1111
dispatchHandlerEvent	1112
~IWindowNotifyHandler	1112

The IWindowNotifyHandler class is the base class for all window notification handlers.

Public Functions

Constructors

You can construct and destruct objects of this class.

IWindowNotifyHandler

Provides the default constructor.

Note: Generally you do not need to construct an object of this class. Calling IWindow::enableNotification (p. 1067) causes an IWindowNotifyHandler object to be constructed and added to the window, if necessary.

```
IWindowNotifyHandler();
```

Win	PM	Motif
Y	*Y*	*Y*

© Copyright IBM Corp. 1992, 1996

IWindowNotifyHandler

~IWindowNotifyHandler

```
virtual
  ~IWindowNotifyHandler();
```

Win	PM	Motif
Y	Y	Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	**messageFile**	**setMessageFile**
asString	**messageText**	**version**

Protected Functions

Event Dispatching

Notification handlers process events that are sent or posted to a window by calling observer objects interested in those events.

dispatchHandlerEvent

This function notifies the window observers when it receives any of the following events:

- Size event
- Position event
- Visible event
- Enable or disable event
- Destroy event
- Command event
- System command event

If you create a class derived from IWindowNotifyHandler, its dispatchHandlerEvent function should call IWindowNotifyHandler::dispatchHandlerEvent for events it does not process.

```
virtual Boolean
  dispatchHandlerEvent( IEvent& event );
```

Win	PM	Motif
Y	*Y*	*Y*

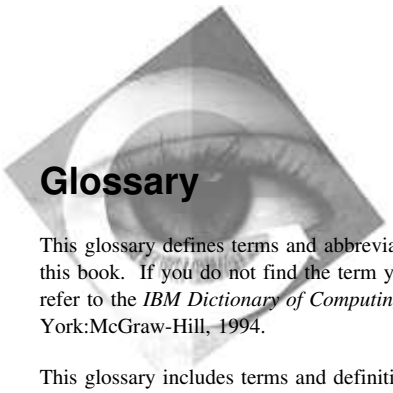
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	**unrecoverable**	

© IBM Corp. 1992, 1996



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 places beside the image. (A system with two sound tracks can have either

© Copyright IBM Corp. 1992, 1996

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

© IBM Corp. 1992, 1996

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. 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 interpetation 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.

© IBM Corp. 1992, 1996

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. 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 kernal 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.

© IBM Corp. 1992, 1996

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. 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.

© IBM Corp. 1992, 1996

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). (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.

© IBM Corp. 1992, 1996

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 (\0). 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).

© IBM Corp. 1992, 1996

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 (::). 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 works 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.

© IBM Corp. 1992, 1996

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. 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 byes 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 adapater.

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 pel 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.

© IBM Corp. 1992, 1996

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 Varlag.

© Copyright IBM Corp. 1992, 1996

Index

Special Characters

~ChildCursor
IWindow::ChildCursor 1100
~Cursor
IAcceleratorTable::Cursor 39
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 102
IClipboard::Cursor 168
IMenu::Cursor 551
IProfile::Cursor 739
ISubmenu::Cursor 931
ITextSpinButton::Cursor 974
IThread::Cursor 1002
ITimer::Cursor 1019
~ExceptionFn
IWindow::ExceptionFn 1103
~I3StateCheckBox
I3StateCheckBox 10
~IAccelerator
IAccelerator 20
~IAcceleratorKey
IAcceleratorKey 27
~IAcceleratorTable
IAcceleratorTable 36
~IApplication
IApplication 50
~IBaseComboBox
IBaseComboBox 56
~IBaseListBox
IBaseListBox 85
~IBaseSpinButton
IBaseSpinButton 110
~IBidiSettings
IBidiSettings 127
~IBitmapHandle
IBitmapHandle 132
~IButton
IButton 136
~IButtonNotifyHandler
IButtonNotifyHandler 143
~ICheckBox
ICheckBox 148
~IClipboard
IClipboard 163
~IClipboardHandler
~IClipboardHandler *(continued)*
IClipboardHandler 171
~IColor
IColor 177
~IComboBox
IComboBox 189
~IComboBoxNotifyHandler
IComboBoxNotifyHandler 197
~ICommandConnectionTo
ICommandConnectionTo 207
~ICommandEvent
ICommandEvent 212
~ICommandHandler
ICommandHandler 216
~IControl
IControl 221
~IControlEvent
IControlEvent 228
~ICritSec
ICritSec 235
~ICurrentApplication
ICurrentApplication 241
~IDeviceColor
IDeviceColor 254
~IDrawItemEvent
IDrawItemEvent 259
~IDynamicLinkLibrary
IDynamicLinkLibrary 262
~IEditHandler
IEditHandler 268
~IEntryField
IEntryField 279
~IEntryFieldNotifyHandler
IEntryFieldNotifyHandler 299
~IEvent
IEvent 307
~IFocusHandler
IFocusHandler 323
~IFrameEvent
IFrameEvent 327
~IFrameExtension
IFrameExtension 331
~IFrameExtensions
IFrameExtensions 338
~IFrameFormatEvent
IFrameFormatEvent 340

© Copyright IBM Corp. 1992, 1996

~IFrameHandler
IFrameHandler 344
~IFrameWindow
IFrameWindow 361
~IFrameWindowNotifyHandler
IFrameWindowNotifyHandler 394
~IGroupBox
IGroupBox 398
~IGUIColor
IGUIColor 406
~IHandler
IHandler 412
~IHelpErrorEvent
IHelpErrorEvent 416
~IHelpHandler
IHelpHandler 422
~IHelpHyperlinkEvent
IHelpHyperlinkEvent 430
~IHelpMenuBarEvent
IHelpMenuBarEvent 432
~IHelpNotifyEvent
IHelpNotifyEvent 435
~IHelpSubitemNotFoundEvent
IHelpSubitemNotFoundEvent 437
~IHelpTutorialEvent
IHelpTutorialEvent 440
~IHelpWindow
IHelpWindow 448
~IKeyboardConnectionTo
IKeyboardConnectionTo 478
~IKeyboardEvent
IKeyboardEvent 483
~IKeyboardHandler
IKeyboardHandler 492
~IListBox
IListBox 501
~IListBoxDrawItemEvent
IListBoxDrawItemEvent 510
~IListBoxDrawItemHandler
IListBoxDrawItemHandler 514
~IListBoxNotifyHandler
IListBoxNotifyHandler 518
~IListBoxSizeItemEvent
IListBoxSizeItemEvent 521
~IMenu
IMenu 536
~IMenuBar
IMenuBar 555
~IMenuDrawItemEvent
IMenuDrawItemEvent 562
~IMenuDrawItemHandler
IMenuDrawItemHandler 566
~IMenuEvent
IMenuEvent 571
~IMenuHandler
IMenuHandler 577
~IMenuItem
IMenuItem 585
~IMenuNotifyHandler
IMenuNotifyHandler 602
~IMessageBox
IMessageBox 605
~IMouseClickEvent
IMouseClickEvent 621
~IMouseConnectionTo
IMouseConnectionTo 625
~IMouseEvent
IMouseEvent 629
~IMouseHandler
IMouseHandler 632
~IMousePointerEvent
IMousePointerEvent 636
~IMousePointerHandler
IMousePointerHandler 639
~IMultiLineEdit
IMultiLineEdit 648
~IMultiLineEditNotifyHandler
IMultiLineEditNotifyHandler 670
~INumericSpinButton
INumericSpinButton 674
~INumericSpinButtonNotifyHandler
INumericSpinButtonNotifyHandler 683
~IObjectWindow
IObjectWindow 687
~IOutlineBox
IOutlineBox 691
~IPaintConnectionTo
IPaintConnectionTo 699
~IPaintEvent
IPaintEvent 702
~IPaintHandler
IPaintHandler 707
~IPointerHandle
IPointerHandle 710
~IPopUpMenu
IPopUpMenu 714
~IPrivateResource
IPrivateResource 720
~IProcedureAddress
IProcedureAddress 725

© IBM Corp. 1992, 1996

~IProfile
IProfile 731
~IProgressIndicator
IProgressIndicator 750
~IProgressIndicatorNotifyHandler
IProgressIndicatorNotifyHandler 771
~IPushButton
IPushButton 777
~IRadioButton
IRadioButton 788
~IRecoordHandler
IRecoordHandler 797
~IResizeEvent
IResizeEvent 800
~IResizeHandler
IResizeHandler 803
~IResource
IResource 805
~IResourceLibrary
IResourceLibrary 812
~IResourceLock
IResourceLock 821
~IScrollBar
IScrollBar 827
~IScrollBarNotifyHandler
IScrollBarNotifyHandler 839
~IScrollEvent
IScrollEvent 842
~IScrollHandler
IScrollHandler 846
~ISelectHandler
ISelectHandler 852
~ISettingButton
ISettingButton 858
~ISettingButtonNotifyHandler
ISettingButtonNotifyHandler 864
~ISharedResource
ISharedResource 867
~IShowListHandler
IShowListHandler 871
~ISlider
ISlider 880
~ISliderArmHandler
ISliderArmHandler 889
~ISliderDrawHandler
ISliderDrawHandler 893
~ISpinHandler
ISpinHandler 897
~IStaticText
IStaticText 904
~ISubmenu
ISubmenu 923
~ISWP
ISWP 935
~ISystemBitmapHandle
ISystemBitmapHandle 942
~ISystemMenu
ISystemMenu 947
~ISystemPointerHandle
ISystemPointerHandle 952
~ITextControl
ITextControl 956
~ITextControlNotifyHandler
ITextControlNotifyHandler 961
~ITextSpinButton
ITextSpinButton 965
~ITextSpinButtonNotifyHandler
ITextSpinButtonNotifyHandler 978
~IThread
IThread 985
~IThreadFn
IThreadFn 1005
~IThreadMemberFn
IThreadMemberFn 1011
~ITimer
ITimer 1015
~ITimerFn
ITimerFn 1022
~ITitle
ITitle 1034
~ITitleNotifyHandler
ITitleNotifyHandler 1041
~IWindow
IWindow 1058
~IWindowNotifyHandler
IWindowNotifyHandler 1112

Numerics

I3StateCheckBox 8
I3StateCheckBox::Style 16

A

abortRetryIgnoreButton
IMessageBox 608
IAccelerator 17
IFrameWindow 384
acceleratorHandle
IWindow 1047

IAcceleratorKey 22
IAcceleratorTable 30
IWindow 1047
IAcceleratorTable::Cursor 39
IAccelTblHandle 42
actionType
IAcceleratorKey 23
activated
IFrameHandler 344
activateId
IFrameWindow 384
activeColor
ITitle 1031
IWindow 1052
activeColorId
IWindow 1093
activeTextBackgroundColor
ITitle 1031
activeTextBackgroundColorId
ITitle 1038
activeTextForegroundColor
ITitle 1031
activeTextForegroundColorId
ITitle 1039
add
IComboBox 183
IListBox 496
IMenu 536
IMultiLineEdit 658
ISubmenu 924
ITextSpinButton 968
addAscending
IComboBox 184
IListBox 497
addAsFirst
IComboBox 185
IListBox 498
ITextSpinButton 968
addAsLast
IComboBox 186
IListBox 499
IMultiLineEdit 658
ITextSpinButton 969
addAsNext
IComboBox 187
IListBox 499
IMenu 536
ISubmenu 924
ITextSpinButton 969
addAtOffset
addAtOffset *(continued)*
IMultiLineEdit 659
addBitmap
IMenu 527
ISubmenu 920
addBorder
IBaseSpinButton 108
IPushButton 775
addDefaultHandler
IFrameWindow 379
addDescending
IComboBox 187
IListBox 500
addDetent
ISlider 880
addExtension
IFrameWindow 361
addHandler
IWindow 1085
addId
IComboBox 193
IListBox 505
addItem
IMenu 529
ISubmenu 921
addKey
IAcceleratorTable 31
addLibraries
IHelpWindow 448
addLine
IMultiLineEdit 650
addLineAsLast
IMultiLineEdit 650
addObserver
IWindow 1091
addOrReplaceElementWithKey
IProfile 733
addOrReplaceKey
IAcceleratorTable 32
addRelatedHandleToWindowSet
IWindow 1081
addSeparator
IMenu 529
ISubmenu 922
addSourceEmphasis
IMenuHandler 580
addSubmenu
IMenu 530
addText
IMenu 531

© IBM Corp. 1992, 1996

addText *(continued)*
ISubmenu 922
addToWindowList
IFrameWindow 376
addToWindowSet
IWindow 1082
adjustPriority
IApplication 47
IThread 997
Advanced Control, Dialog, and Handler Classes xxii
overview xxii
alignBottom
IProgressIndicator 764
alignCentered
IProgressIndicator 764
alignLeft
IProgressIndicator 764
alignment
IBaseSpinButton 107
IEntryField 272
IProgressIndicator 745
IStaticText 901
alignNoAdjust
IFrameWindow 384
alignRight
IProgressIndicator 764
alignTop
IProgressIndicator 765
allowsMouseClickFocus
IButton 136
alt
IKey 467
anchorBlock
ICurrentThread 246
IAnchorBlockHandle 44
animated
IFrameWindow 385
anyData
IBaseComboBox 70
IEntryField 291
appContext
ICurrentThread 246
appDBCSStatus
IFrameWindow 385
IApplication 46
applicationModal
IMessageBox 608
applicationOrientation
ICoordinateSystem 232
applicationOrKeyAt
IProfile 731
apply
IBidiSettings 126
appShell
ICurrentThread 246
argc
ICurrentApplication 237
argv
ICurrentApplication 237
armChangeId
IProgressIndicator 764
armPixelOffset
IProgressIndicator 746
armRange
IProgressIndicator 746
armSize
ISlider 876
armTickOffset
IProgressIndicator 746
armTrackId
ISlider 883
arrowDown
ISpinHandler 898
arrowUp
ISpinHandler 898
asACCEL
IAcceleratorKey 27
asDebugInfo
IApplication 46
ICurrentApplication 238
IDynamicLinkLibrary 263
IHandle 409
IHandler 412
IResourceId 809
IResourceLibrary 812
IThread 986
ITimer 1015
IWindow 1059
asIndex
IBaseComboBox::Cursor 77
IBaseListBox::Cursor 103
asLong
IEventData 317
asPixel
IColor 178
asRGBLong
IColor 178
asString
IApplication 47

asString *(continued)*
IDynamicLinkLibrary 263
IHandle 409
IHandler 413
IResourceId 809
IResourceLibrary 812
IThread 986
ITimer 1015
IWindow 1059
asUnsigned
IHandle 409
asUnsignedLong
IEventData 317
atEnd
IMenuItem 594
attachClient
IFrameWindow 377
attachedToId
IFrameExtension 332
attachTo
IFrameExtension 332
attribute
IMenuItem 592
autoInitGUI
IThread 986
autoScroll
IBaseComboBox 70
IEntryField 291
autoSelect
I3StateCheckBox 14
ICheckBox 152
IRadioButton 793
autoSize
IScrollBar 835
autoTab
IEntryField 292

B

backgroundColor
IBaseListBox 83
IBaseSpinButton 109
IButton 134
IEntryField 277
IFrameWindow 356
IMenu 532
IMultiLineEdit 646
IProgressIndicator 747
IStaticText 902
IWindow 1052
backgroundColorId
IWindow 1093
backgroundFrame
IOutlineBox 694
IBaseComboBox 52
IBaseComboBox::Cursor 75
IBaseComboBox::Style 79
IBaseListBox 81
IBaseListBox::Cursor 101
IBaseListBox::Style 105
baseRectFor
IFrameExtension 333
IBaseSpinButton 106
IBaseSpinButton::Style 121
beginFlashing
IFrameWindow 375
behind
ISWP 938
IBidiSettings 122
bitmap
IClipboard 156
IMenuItem 587
bitmapFormat
IClipboard 164
IBitmapHandle 131
blueMix
IColor 174
border
IFrameWindow 385
IMultiLineEdit 664
border3D
IBaseComboBox 70
IBaseListBox 98
IBaseSpinButton 117
IEntryField 292
IMultiLineEdit 665
IOutlineBox 694
IProgressIndicator 765
IStaticText 912
borderColor
ITitle 1031
IWindow 1052
borderColorId
IWindow 1093
borderHeight
IFrameWindow 353
borderSize
IFrameWindow 353
borderWidth
IFrameWindow 353

© IBM Corp. 1992, 1996

bottom
IStaticText 912
IButton 134
IButton::Style 142
buttonClickId
IButton 139
IButtonNotifyHandler 143
buttonsBottom
ISlider 883
buttonSeparator
IMenuItem 595
buttonsLeft
ISlider 884
buttonsPosition
ISlider 877
buttonsRight
ISlider 884
buttonsTop
ISlider 884

C

calcLimitSize
IStaticText 910
calcMinimumSize
I3StateCheckBox 13
IBaseComboBox 68
IBaseListBox 96
IBaseSpinButton 116
ICheckBox 151
IEntryField 289
IGroupBox 401
IProgressIndicator 763
IPushButton 781
IRadioButton 792
IScrollBar 835
IStaticText 910
IWindow 1091
calcRect
IFrameHandler 344
cancelButton
IMessageBox 608
capturePointer
IWindow 1071
center
IStaticText 912
centerAlign
IBaseSpinButton 117
IEntryField 292
changeCount
IBaseComboBox 67
IBaseListBox 96
char1
IEventData 315
char2
IEventData 315
char3
IEventData 315
char4
IEventData 315
character
IAcceleratorKey 27
IKeyboardEvent 482
characterKeyPress
IKeyboardConnectionTo 479
IKeyboardHandler 493
characterSize
IWindow 1063
characterTypeId
IEntryField 290
charType
IEntryField 273
ICheckBox 146
ICheckBox::Style 154
checked
IMenuItem 593
checkItem
IMenu 543
ISubmenu 927
childAt
IWindow 1052
ChildCursor
IWindow::ChildCursor 1100
classDefaultAttribute
IMenuItem 594
classDefaultStyle
I3StateCheckBox 14
IBaseComboBox 70
IBaseListBox 98
ICheckBox 152
IComboBox 193
IEntryField 292
IFrameWindow 385
IGroupBox 402
IHelpWindow 456
IListBox 505
IMenu 548
IMenuBar 558
IMenuItem 595

classDefaultStyle *(continued)*
IMultiLineEdit 665
INumericSpinButton 679
IOutlineBox 694
IProgressIndicator 765
IPushButton 782
IRadioButton 793
IScrollBar 836
ISlider 884
IStaticText 912
ITextSpinButton 972
Classes
I3StateCheckBox 8
I3StateCheckBox::Style 16
IAccelerator 17
IAcceleratorKey 22
IAcceleratorTable 30
IAcceleratorTable::Cursor 39
IAccelTblHandle 42
IAnchorBlockHandle 44
IApplication 46
IBaseComboBox 52
IBaseComboBox::Cursor 75
IBaseComboBox::Style 79
IBaseListBox 81
IBaseListBox::Cursor 101
IBaseListBox::Style 105
IBaseSpinButton 106
IBaseSpinButton::Style 121
IBidiSettings 122
IBitmapHandle 131
IButton 134
IButton::Style 142
IButtonNotifyHandler 143
ICheckBox 146
ICheckBox::Style 154
IClipboard 155
IClipboard::Cursor 167
IClipboardHandler 170
IColor 174
IComboBox 182
IComboBox::Style 196
IComboBoxNotifyHandler 197
ICommand 200
ICommandConnectionTo 206
ICommandEvent 210
ICommandHandler 214
IContextHandle 219
IControl 221
IControl::Style 226

Classes *(continued)*
IControlEvent 227
ICoordinateSystem 230
ICritSec 234
ICurrentApplication 237
ICurrentThread 243
IDeviceColor 253
IDisplayHandle 256
IDrawItemEvent 258
IDynamicLinkLibrary 261
IEditHandler 267
IEntryField 271
IEntryField::Style 297
IEntryFieldNotifyHandler 299
IEnumHandle 302
IEvent 304
IEventData 313
IEventParameter1 319
IEventParameter2 320
IEventResult 321
IFocusHandler 322
IFrameEvent 326
IFrameExtension 329
IFrameExtensions 338
IFrameFormatEvent 339
IFrameHandler 342
IFrameWindow 349
IFrameWindow::Style 392
IFrameWindowNotifyHandler 393
IGroupBox 396
IGroupBox::Style 404
IGUIColor 405
IHandle 408
IHandler 411
IHelpErrorEvent 416
IHelpHandler 420
IHelpHyperlinkEvent 429
IHelpMenuBarEvent 432
IHelpNotifyEvent 434
IHelpSubitemNotFoundEvent 437
IHelpTutorialEvent 440
IHelpWindow 442
IHelpWindow::Settings 459
IHelpWindow::Style 464
IHighEventParameter 465
IKey 466
IKey::KeyModifier 475
IKeyboardConnectionTo 477
IKeyboardEvent 481
IKeyboardHandler 490

© IBM Corp. 1992, 1996

Classes *(continued)*
IListBox 495
IListBox::Style 508
IListBoxDrawItemEvent 509
IListBoxDrawItemHandler 513
IListBoxNotifyHandler 518
IListBoxSizeItemEvent 521
ILowEventParameter 524
IMenu 525
IMenu::Cursor 550
IMenu::Style 553
IMenuBar 554
IMenuBar::Style 561
IMenuDrawItemEvent 562
IMenuDrawItemHandler 565
IMenuDrawItemHandler::DrawFlag 570
IMenuEvent 571
IMenuHandle 574
IMenuHandler 576
IMenuItem 582
IMenuItem::Attribute 599
IMenuItem::Style 600
IMenuNotifyHandler 601
IMessageBox 604
IMessageBox::Style 613
IMessageQueueHandle 615
IModuleHandle 618
IMouseClickEvent 620
IMouseConnectionTo 624
IMouseEvent 628
IMouseHandler 631
IMousePointerEvent 635
IMousePointerHandler 638
IMultiLineEdit 641
IMultiLineEdit::Style 668
IMultiLineEditNotifyHandler 669
INumericSpinButton 672
INumericSpinButton::Style 682
INumericSpinButtonNotifyHandler 683
IObjectWindow 686
IOutlineBox 689
IOutlineBox::Style 697
IPaintConnectionTo 698
IPaintEvent 701
IPaintHandler 706
IPointerHandle 709
IPopUpMenu 712
IPresSpaceHandle 717
IPrivateResource 719
IProcedureAddress 723

Classes *(continued)*
IProcessId 727
IProfile 729
IProfile::Cursor 738
IProfileHandle 741
IProgressIndicator 743
IProgressIndicator::Style 770
IProgressIndicatorNotifyHandler 771
IPushButton 774
IPushButton::Style 785
IRadioButton 786
IRadioButton::Style 795
IRecoordHandler 796
IResizeEvent 800
IResizeHandler 802
IResource 805
IResourceId 808
IResourceLibrary 811
IResourceLock 820
IScrollBar 823
IScrollBar::Style 838
IScrollBarNotifyHandler 839
IScrollEvent 842
IScrollHandler 845
ISelectHandler 851
ISemaphoreHandle 855
ISettingButton 857
ISettingButtonNotifyHandler 863
ISharedResource 866
IShowListHandler 870
ISlider 874
ISlider::Style 887
ISliderArmHandler 888
ISliderDrawHandler 892
ISpinHandler 896
IStaticText 900
IStaticText::Style 916
IStringHandle 917
ISubmenu 919
ISubmenu::Cursor 931
ISWP 934
ISWPArray 940
ISystemBitmapHandle 942
ISystemMenu 946
ISystemPointerHandle 952
ITextControl 955
ITextControlNotifyHandler 960
ITextSpinButton 963
ITextSpinButton::Cursor 974
ITextSpinButton::Style 977

Classes *(continued)*
ITextSpinButtonNotifyHandler 978
IThread 981
IThread::Cursor 1001
IThreadFn 1004
IThreadHandle 1006
IThreadId 1008
IThreadMemberFn 1010
ITimer 1013
ITimer::Cursor 1019
ITimerFn 1021
ITimerMemberFn 1023
ITimerMemberFn0 1026
ITitle 1029
ITitleNotifyHandler 1041
IWindow 1044
IWindow::ChildCursor 1100
IWindow::ExceptionFn 1103
IWindow::Style 1105
IWindowHandle 1107
IWindowNotifyHandler 1111
clear
IEntryField 275
IMultiLineEdit 644
clearBackground
IPaintEvent 702
clearLock
IResourceLock 821
click
IButton 137
client
IFrameWindow 355
clientHandle
IFrameWindow 355
clientRect
IFrameFormatEvent 340
clientRectFor
IFrameWindow 368
IClipboard 155
IClipboard::Cursor 167
clipboardEmptied
IClipboardHandler 172
IClipboardHandler 170
clipboardHasTextFormat
ITextControl 955
clipChildren
IWindow 1096
clipSiblings
IWindow 1097
clipToParent
clipToParent *(continued)*
IWindow 1097
close
IClipboard 161
IDynamicLinkLibrary 264
IFrameWindow 371
IObjectWindow 687
closed
IFrameHandler 345
closeId
IFrameWindow 384
CMDSRC_MENU 583, 712
IColor 174
IWindow 1085
IComboBox 182
IComboBox::Style 196
IComboBoxNotifyHandler 197
ICommand 200
ICommandConnectionTo 208
ICommandHandler 217
IEntryField 292
IFrameHandler 345
ICommandConnectionTo 206
ICommandEvent 210
ICommandHandler 214
commandId
IAcceleratorKey 23
ICommandEvent 211
IHelpMenuBarEvent 433
IWindow 1093
commandType
IMenuItem 584
communicationWindow
IHelpWindow 453
Compound Document Framework Classes xxii
overview xxii
containsApplication
IProfile 736
containsKeyLike
IAcceleratorTable 37
containsKeyName
IProfile 736
contentsWindow
IHelpWindow 453
IContextHandle 219
IControl 221
IFrameExtension 335
IControl::Style 226
IControlEvent 227
controlHandle

© IBM Corp. 1992, 1996

controlHandle *(continued)*
IEvent 309
controlId
IControlEvent 228
IHelpNotifyEvent 435
controlSelect
IHelpHandler 424
controlWindow
IEvent 310
convertToApplication
ICoordinateSystem 231
convertToGUIStyle
I3StateCheckBox 11
IBaseComboBox 63
IBaseListBox 91
IBaseSpinButton 113
ICheckBox 149
IEntryField 284
IFrameWindow 370
IGroupBox 399
IMenu 546
IMenuBar 556
IMenuItem 590
IMultiLineEdit 657
INumericSpinButton 677
IOutlineBox 692
IProgressIndicator 756
IPushButton 778
IRadioButton 790
IScrollBar 831
ISlider 881
IStaticText 906
ITextSpinButton 967
IWindow 1074
convertToNative
ICoordinateSystem 231
ICoordinateSystem 230
copy
IEntryField 275
IMultiLineEdit 644
count
IBaseComboBox 56
IBaseListBox 85
ICritSec 235
coverPageWindow
IHelpWindow 453
cpp files, description xxiii
CPP.NDX xxiv
CPP*.INF xxiv
CPPBRS.NDX xxiv
CPPWO*.DEF xxiv
CPPWO*.DLL xxiv
CPPWO*.LIB xxiv
CPPWO*.RSP xxiv
CPPWOC3.LIB xxiv
CPPWOC3U.MSG xxiv
create
IFrameWindow 379
IWindow 1086
ICritSec 234
ctrl
IKey 467
current
IApplication 48
IThread 995
ICurrentApplication 237
currentHandle
IThread 995
currentId
IThread 995
currentPID
IApplication 49
ICurrentThread 243
Cursor
IAcceleratorTable::Cursor 39
IBaseComboBox::Cursor 75
IBaseListBox::Cursor 101
IClipboard::Cursor 167
IMenu 536
IMenu::Cursor 550
IProfile::Cursor 738
ISubmenu 924
ISubmenu::Cursor 931
ITextSpinButton::Cursor 974
IThread::Cursor 1001
ITimer::Cursor 1019
cursorLinePosition
IMultiLineEdit 651
cursorPosition
IEntryField 282
IMultiLineEdit 651
cut
IEntryField 276
IMultiLineEdit 645
cx
ISWP 938
cy
ISWP 938

D

data
  IClipboard 157
data member names, conventions xxv
dataUpdateId
  IEntryField 290
  IMultiLineEdit 664
dbcsData
  IBaseComboBox 70
  IEntryField 292
DDE4C01E.MSG xxiv
deactivated
  IFrameHandler 345
deactivateId
  IFrameWindow 384
defaultApplicationName
  IProfile 731
defaultAttribute
  IMenuItem 584
defaultAutoInitGUI
  IThread 986
defaultButton
  IPushButton 782
defaultHandler
  IMousePointerHandler 639
  IRecoordHandler 797
defaultMousePointer
  IMousePointerEvent 636
defaultOrdering
  IFrameWindow 369
  IWindow 1073
defaultProcedure
  IHandler 415
  IWindow 1082
defaultPushButton
  IFrameWindow 365
  IWindow 1064
defaultQueueSize
  IThread 988
defaultStackSize
  IThread 990
defaultStyle
  I3StateCheckBox 11
  IBaseComboBox 63
  IBaseListBox 91
  ICheckBox 149
  IComboBox 190
  IEntryField 284
  IFrameWindow 370
defaultStyle *(continued)*
  IGroupBox 399
  IHelpWindow 451
  IListBox 503
  IMenu 546
  IMenuBar 556
  IMenuItem 591
  IMultiLineEdit 657
  INumericSpinButton 677
  IOutlineBox 692
  IProgressIndicator 756
  IPushButton 778
  IRadioButton 790
  IScrollBar 831
  ISlider 881
  IStaticText 906
  ITextSpinButton 967
defButton1
  IMessageBox 608
defButton2
  IMessageBox 609
defButton3
  IMessageBox 609
deferCreation
  IFrameWindow 383
deleteAt
  IMenu 537
  ISubmenu 924
deleteElementWithApplication
  IProfile 734
deleteElementWithKey
  IProfile 734
deleteId
  IWindow 1093
deleteIsInProcess
  IWindow 1082
deleteItem
  IMenu 541
  ISubmenu 925
deleteProfile
  IProfile 734
deselect
  IBaseComboBox 61
  IBaseListBox 89
  IMultiLineEdit 664
  ISettingButton 859
deselectAll
  IBaseComboBox 62
  IBaseListBox 90
deselectItem

© IBM Corp. 1992, 1996

deselectItem *(continued)*
IListBoxDrawItemHandler 516
desktopWindow
IWindow 1049
destroyHandle
IAcceleratorTable 36
detentPosition
ISlider 880
IDeviceColor 253
IDeviceColor 253
dialogBackground
IFrameWindow 385
dialogBorder
IFrameWindow 385
dialogControls
IThread 988
Direct Manipulation Classes xxii
overview xxii
disable
IEntryField 281
IHandler 413
IWindow 1074
disableAutoScroll
IEntryField 274
disableAutoSelect
I3StateCheckBox 9
ICheckBox 147
IRadioButton 787
ISettingButton 858
disableAutoTab
IEntryField 274
disableCommand
IEntryField 278
disableCursorSelect
IRadioButton 789
disabled
IMenuItem 594
IWindow 1097
disableDataUpdate
IBaseSpinButton 113
IEntryField 282
IMultiLineEdit 653
disabledBackgroundColor
IFrameWindow 356
IMenu 532
IWindow 1052
disabledBackgroundColorId
IWindow 1094
disableDefault
IPushButton 777
disabledForegroundColor
IButton 135
IMenu 533
IWindow 1053
disabledForegroundColorId
IWindow 1094
disableDrawItem
IBaseListBox 92
IProgressIndicator 751
disableExtendedSelect
IBaseListBox 92
disableFastSpin
IBaseSpinButton 110
disableFillBackground
IStaticText 904
disableGroup
IControl 222
disableHalftone
IStaticText 907
disableHelp
IPushButton 777
disableInsertMode
IEntryField 279
disableItem
IMenu 544
ISubmenu 927
disableMargin
IEntryField 280
disableMinimumSizeCaching
IWindow 1064
disableMouseClickFocus
IButton 137
disableMultipleSelect
IBaseListBox 92
disableNoAdjustPosition
IBaseListBox 92
disableNotification
IWindow 1067
disableRibbonStrip
IProgressIndicator 753
disableSnapToTick
IProgressIndicator 755
disableStrikeout
IStaticText 907
disableSymmetricSwapping
IBidiSettings 122
disableSystemCommand
IPushButton 779
disableTabStop
IControl 222

disableUnderscore
IStaticText 907
disableUpdate
IMultiLineEdit 653
IWindow 1074
disableWordByWordReordering
IBidiSettings 123
disableWordWrap
IMultiLineEdit 661
discard
IEntryField 276
IMultiLineEdit 645
dismiss
IFrameWindow 366
dispatch
IWindow 1082
dispatchHandlerEvent
IButtonNotifyHandler 144
IClipboardHandler 171
IComboBoxNotifyHandler 198
ICommandHandler 217
IEditHandler 269
IEntryFieldNotifyHandler 300
IFocusHandler 324
IFrameHandler 347
IFrameWindowNotifyHandler 394
IHandler 414
IHelpHandler 424
IKeyboardHandler 492
IListBoxDrawItemHandler 516
IListBoxNotifyHandler 519
IMenuDrawItemHandler 567
IMenuHandler 578
IMenuNotifyHandler 602
IMouseHandler 633
IMultiLineEditNotifyHandler 670
INumericSpinButtonNotifyHandler 684
IPaintHandler 708
IProgressIndicatorNotifyHandler 772
IRecoordHandler 798
IResizeHandler 804
IScrollBarNotifyHandler 840
IScrollHandler 847
ISelectHandler 853
ISettingButtonNotifyHandler 864
IShowListHandler 872
ISliderArmHandler 890
ISliderDrawHandler 894
ISpinHandler 898
ITextControlNotifyHandler 961
dispatchHandlerEvent *(continued)*
ITextSpinButtonNotifyHandler 979
ITitleNotifyHandler 1042
IWindowNotifyHandler 1112
dispatchingWindow
IEvent 310
dispatchRemainingHandlers
IWindow 1061
displayBitmapFormat
IClipboard 164
IDisplayHandle 256
displayMetafileFormat
IClipboard 165
displaySize
ITextControl 957
displayTextFormat
IClipboard 165
draw
IFrameHandler 345
IMenuDrawItemHandler 567
drawArm
ISliderDrawHandler 894
drawBackground
ISliderDrawHandler 895
drawItem
IBaseListBox 98
IListBoxDrawItemHandler 516
IMenuItem 595
IDrawItemEvent 258
drawRibbonStrip
ISliderDrawHandler 895
drawSeparator
IFrameExtension 335
drawShaft
ISliderDrawHandler 895
drawText
IPaintEvent 703
drewChecked
IMenuDrawItemHandler 569
drewDisabled
IMenuDrawItemHandler 569
drewSelected
IMenuDrawItemHandler 569
dropDownType
IBaseComboBox 70
Dynamic Data Exchange Classes xxii
overview xxii
IDynamicLinkLibrary 261

© IBM Corp. 1992, 1996

E

edit
IEditHandler 269
IEditHandler 267
editRegionHeight
IMultiLineEdit 655
editRegionWidth
IMultiLineEdit 655
element
IAcceleratorTable::Cursor 40
elementAt
IBaseComboBox 57
IBaseListBox 85
IMenu 537
ISubmenu 924
ITextSpinButton 969
elementWithKey
IProfile 735
empty
IClipboard 162
IMenuBar 559
enable
IBaseSpinButton 112
IEntryField 282
IHandler 413
IWindow 1074
enableAutoScroll
IEntryField 274
enableAutoSelect
I3StateCheckBox 9
ICheckBox 147
IRadioButton 787
ISettingButton 858
enableAutoTab
IEntryField 274
enableCommand
IEntryField 278
enableCursorSelect
IRadioButton 789
enableDataUpdate
IBaseSpinButton 113
IEntryField 282
IMultiLineEdit 653
enableDefault
IPushButton 777
enableDrawItem
IBaseListBox 92
IProgressIndicator 751
enableExtendedSelect
IBaseListBox 92
enableFastSpin
IBaseSpinButton 111
enableFillBackground
IStaticText 904
enableGroup
IControl 222
enableHalftone
IStaticText 907
enableHelp
IPushButton 777
enableId
IWindow 1094
enableInsertMode
IEntryField 280
enableItem
IMenu 544
ISubmenu 927
enableMargin
IEntryField 281
enableMinimumSizeCaching
IWindow 1064
enableMouseClickFocus
IButton 137
enableMultipleSelect
IBaseListBox 93
enableNoAdjustPosition
IBaseListBox 93
enableNotification
IBaseComboBox 60
IBaseListBox 88
IButton 137
IEntryField 281
IFrameWindow 369
IMenu 540
IMultiLineEdit 652
INumericSpinButton 675
IProgressIndicator 752
IScrollBar 829
ISettingButton 859
ITextSpinButton 966
ITitle 1034
IWindow 1067
enableRibbonStrip
IProgressIndicator 753
enableSnapToTick
IProgressIndicator 755
enableStrikeout
IStaticText 907

enableSymmetricSwapping
IBidiSettings 123
enableSystemCommand
IPushButton 779
enableTabStop
IControl 223
enableUnderscore
IStaticText 907
enableUpdate
IMultiLineEdit 653
IWindow 1075
enableWordByWordReordering
IBidiSettings 123
enableWordWrap
IMultiLineEdit 661
end
IEntryField 291
IMultiLineEdit 664
endFlashing
IFrameWindow 375
enter
ISelectHandler 853
enterButton
IMessageBox 609
enterCancelButton
IMessageBox 609
enterId
IBaseComboBox 69
IBaseListBox 97
IEntryField 271
IEntryField::Style 297
IEntryFieldNotifyHandler 299
IEnumHandle 302
error
IHelpErrorEvent 417
errorIcon
IMessageBox 609
IEvent 304
IEventData 313
eventId
IEvent 307
IEventParameter1 319
IEventParameter2 320
IEventResult 321
eventType
IEvent 308
exceptionFunction
IWindow 1062
exit
ICurrentApplication 240
exit *(continued)*
ICurrentThread 244
exportSelectedTextToFile
IMultiLineEdit 649
exportToFile
IMultiLineEdit 649
extendedSelect
IBaseListBox 98
extendedStyle
IMenuItem 592
IWindow 1087
extensions
IFrameWindow 378

F

fastSpin
IBaseSpinButton 117
fileName
IDynamicLinkLibrary 263
IResourceLibrary 813
fillBackground
IStaticText 912
fillBackgroundId
IStaticText 910
fillColor
IStaticText 902
fillColorId
IStaticText 911
findExtension
IFrameWindow 379
findExtensions
IFrameWindow 377
first
IBaseComboBox 69
IBaseListBox 97
fixedSize
IFrameExtension 333
fl
ISWP 938
flags
ISWP 936
IFocusHandler 322
focusId
IWindow 1094
font
IWindow 1063
fontId
IWindow 1094

© IBM Corp. 1992, 1996

foregroundColor
IBaseSpinButton 109
IButton 135
IEntryField 277
IGroupBox 397
IMenu 533
IMultiLineEdit 647
IOutlineBox 689
IProgressIndicator 747
IScrollBar 824
IStaticText 902
IWindow 1053
foregroundColorId
IWindow 1094
foregroundFrame
IOutlineBox 695
format
IClipboard 159
IFrameHandler 346
formatAsHandle
IClipboard 160
formatCount
IClipboard 160
frame
IFrameEvent 327
framed
IMenuItem 594
IFrameEvent 326
IFrameExtension 329
IFrameExtensions 338
IFrameFormatEvent 339
IFrameHandler 342
frameRectFor
IFrameWindow 368
IFrameWindow 349
IFrameWindow::Style 392
IFrameWindowNotifyHandler 393
function arguments, conventions xxvi
function return types, conventions xxv

G

global names, conventions xxv
gotFocus
IFocusHandler 324
greenMix
IColor 175
group
IControl 224
IGroupBox 396
IGroupBox::Style 404
IGUIColor 405

H

h files, description xxiii
halftone
IStaticText 912
halftoneFrame
IOutlineBox 695
halftoneId
IStaticText 911
handle 408
IAccelerator 18
IAcceleratorTable 37
ICurrentThread 243
IDynamicLinkLibrary 263
IEvent 311
IHandle 410
IHelpWindow 453
IPrivateResource 721
IProfile 732
IResource 807
IResourceLibrary 813
ISharedResource 868
ISubmenu 925
IThread 996
ITitle 1030
IWindow 1049
handleDrawItem
IProgressIndicator 765
handleError
IHelpHandler 424
handleEventsFor
IHandler 413
IHelpHandler 423
IListBoxDrawItemHandler 515
handleException
IWindow 1062
IWindow::ExceptionFn 1103
handleFor
IFrameWindow 370
IHandler 411
handleWithParent
IWindow 1069
handleWithPointerCaptured
IWindow 1049
hasBitmap
IClipboard 157

hasBorder
IBaseSpinButton 108
IPushButton 775
hasData
IClipboard 157
hasFillBackground
IStaticText 905
hasFocus
IBaseComboBox 54
IBaseSpinButton 107
ISlider 877
IWindow 1049
hasPointerCaptured
IWindow 1072
hasSelectedText
IEntryField 285
IMultiLineEdit 654
hasText
IClipboard 157
hasTextChanged
IEntryField 285
IMultiLineEdit 643
help
IPushButton 782
IHelpErrorEvent 416
IHelpHandler 420
IHelpHyperlinkEvent 429
helpId
IMenuItem 592
IWindow 1063
IHelpMenuBarEvent 432
IHelpNotifyEvent 434
IHelpSubitemNotFoundEvent 437
IHelpTutorialEvent 440
helpUndefined
IHelpHandler 424
IHelpWindow 442
IHelpWindow 453
IHelpWindow::Settings 459
IHelpWindow::Style 464
hide
IGroupBox 400
IHelpWindow 449
IWindow 1075
hideButton
IFrameWindow 386
hideList
IBaseComboBox 60
hidePanelIds
IHelpWindow 450
hideSourceEmphasis
IWindow 1075
IHighEventParameter 465
highHighByte
IEventData 315
highlight
IButton 136
IMenuDrawItemHandler 568
highlighted
IMenuItem 594
highLowByte
IEventData 316
highNumber
IEventData 316
hiliteBackgroundColor
IButton 135
IMenu 533
IWindow 1053
hiliteBackgroundColorId
IWindow 1095
hiliteForegroundColor
IButton 135
IMenu 533
IScrollBar 824
IWindow 1053
hiliteForegroundColorId
IWindow 1095
homeBottom
IProgressIndicator 765
homeLeft
IProgressIndicator 765
homePosition
IProgressIndicator 752
homeRight
IProgressIndicator 766
homeTop
IProgressIndicator 766
horizontal
IProgressIndicator 766
IScrollBar 836
horizontalScroll
IBaseComboBox 71
IBaseListBox 99
IFrameWindow 386
IMultiLineEdit 665
hpp files, description xxiii
hwnd
ISWP 938
hyperlinkSelect
IHelpHandler 425

© IBM Corp. 1992, 1996

I

I3StateCheckBox
I3StateCheckBox 9
IAccelerator
IAccelerator 20
IAcceleratorKey
IAcceleratorKey 25
IAcceleratorTable
IAcceleratorTable 34
IAccelTblHandle
IAccelTblHandle 42
IAnchorBlockHandle
IAnchorBlockHandle 44
IApplication
IApplication 49
IBaseComboBox
IBaseComboBox 55
IBaseListBox
IBaseListBox 84
IBaseSpinButton
IBaseSpinButton 115
IBidiSettings
IBidiSettings 127
IBitmapHandle
IBitmapHandle 131
ibmcl.cat xxv
IButton
IButton 139
IButtonNotifyHandler
IButtonNotifyHandler 143
ICheckBox
ICheckBox 148
IClipboard
IClipboard 163
IClipboardHandler
IClipboardHandler 171
IColor
IColor 176, 180
IComboBox
IComboBox 188
IComboBoxNotifyHandler
IComboBoxNotifyHandler 197
ICommandConnectionTo
ICommandConnectionTo 207
ICommandEvent
ICommandEvent 212
ICommandHandler
ICommandHandler 216
icon
IFrameWindow 352
IContextHandle
IContextHandle 219
IControl
IControl 224
IControlEvent
IControlEvent 228
ICritSec
ICritSec 235
ICurrentApplication
ICurrentApplication 241
ICurrentThread
ICurrentThread 251
id
IApplication 49
ICurrentThread 244
IHelpHyperlinkEvent 430
IMenu 537
IMenuItem 589
IResourceId 810
IThread 996
ITimer 1018
IWindow 1049
idClose
ISystemMenu 949
IDeviceColor
IDeviceColor 253
idHide
ISystemMenu 949
IDisplayHandle
IDisplayHandle 256
idMaximize
ISystemMenu 949
idMinimize
ISystemMenu 949
idMove
ISystemMenu 949
idPulldown
ISystemMenu 950
IDrawItemEvent
IDrawItemEvent 258
idRestore
ISystemMenu 950
idSize
ISystemMenu 950
idWindowList
ISystemMenu 950
IDynamicLinkLibrary
IDynamicLinkLibrary 261

IEditHandler
IEditHandler 269
IEntryField
IEntryField 278, 288
IEntryFieldNotifyHandler
IEntryFieldNotifyHandler 299
IEnumHandle
IEnumHandle 302
IEvent
IEvent 306
IEventData
IEventData 314
IFocusHandler
IFocusHandler 323
IFrameEvent
IFrameEvent 326
IFrameExtension
IFrameExtension 330
IFrameExtensions
IFrameExtensions 338
IFrameFormatEvent
IFrameFormatEvent 339
IFrameHandler
IFrameHandler 344
IFrameWindow
IFrameWindow 357
IFrameWindowNotifyHandler
IFrameWindowNotifyHandler 393
ignoreTab
IMultiLineEdit 665
IGroupBox
IGroupBox 397
IGUIColor
IGUIColor 406
IHandle
IHandle 409
IHandler
IHandler 412
IHelpErrorEvent
IHelpErrorEvent 416
IHelpHandler
IHelpHandler 422
IHelpHyperlinkEvent
IHelpHyperlinkEvent 429
IHelpMenuBarEvent
IHelpMenuBarEvent 432
IHelpNotifyEvent
IHelpNotifyEvent 435
IHelpSubitemNotFoundEvent
IHelpSubitemNotFoundEvent 437
IHelpTutorialEvent
IHelpTutorialEvent 440
IHelpWindow
IHelpWindow 445
IKeyboardConnectionTo
IKeyboardConnectionTo 478
IKeyboardEvent
IKeyboardEvent 483
IKeyboardHandler
IKeyboardHandler 491
IListBox
IListBox 501
IListBoxDrawItemEvent
IListBoxDrawItemEvent 509
IListBoxDrawItemHandler
IListBoxDrawItemHandler 514
IListBoxNotifyHandler
IListBoxNotifyHandler 518
IListBoxSizeItemEvent
IListBoxSizeItemEvent 521
IMenu
IMenu 536, 547
IMenuBar
IMenuBar 555
IMenuDrawItemEvent
IMenuDrawItemEvent 562
IMenuDrawItemHandler
IMenuDrawItemHandler 566
IMenuEvent
IMenuEvent 571
IMenuHandle
IMenuHandle 574
IMenuHandler
IMenuHandler 577
IMenuItem
IMenuItem 585
IMenuNotifyHandler
IMenuNotifyHandler 601
IMessageBox
IMessageBox 604
IMessageQueueHandle
IMessageQueueHandle 615
IModuleHandle
IModuleHandle 618
IMouseClickEvent
IMouseClickEvent 620
IMouseConnectionTo
IMouseConnectionTo 625
IMouseEvent
IMouseEvent 628

© IBM Corp. 1992, 1996

IMouseHandler
IMouseHandler 632
IMousePointerEvent
IMousePointerEvent 635
IMousePointerHandler
IMousePointerHandler 639
importFromFile
IMultiLineEdit 649
IMultiLineEdit
IMultiLineEdit 647
IMultiLineEditNotifyHandler
IMultiLineEditNotifyHandler 669
inactiveColor
ITitle 1031
IWindow 1053
inactiveColorId
IWindow 1095
inactiveTextBackgroundColor
ITitle 1032
inactiveTextBackgroundColorId
ITitle 1039
inactiveTextForegroundColor
ITitle 1032
inactiveTextForegroundColorId
ITitle 1039
incrementChangeCount
IBaseComboBox 67
IBaseListBox 96
index
IColor 178
IMenuItem 589
indexOf
ISWPArray 940
indexWindow
IHelpWindow 454
informationIcon
IMessageBox 609
initialize
IBaseSpinButton 115
IEntryField 288
IFrameWindow 381
IProgressIndicator 762
initializeGUI
ICurrentThread 247
inl files, description xxiii
insertModeId
IEntryField 291
integerWithKey
IProfile 735
interval
ITimer 1016
INumericSpinButton
INumericSpinButton 673
INumericSpinButtonNotifyHandler
INumericSpinButtonNotifyHandler 683
invalidate
IAcceleratorTable::Cursor 40
IBaseComboBox::Cursor 77
IBaseListBox::Cursor 103
IClipboard::Cursor 168
IMenu::Cursor 551
IProfile::Cursor 739
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
IThread::Cursor 1002
ITimer::Cursor 1019
IWindow::ChildCursor 1101
IObjectWindow
IObjectWindow 687
IOutlineBox
IOutlineBox 690
IPaintConnectionTo
IPaintConnectionTo 699
IPaintEvent
IPaintEvent 702
IPaintHandler
IPaintHandler 707
ipfCompatible
IHelpWindow 456
IPointerHandle
IPointerHandle 709
IPopUpMenu
IPopUpMenu 713
IPresSpaceHandle
IPresSpaceHandle 717
IPrivateResource
IPrivateResource 720
IProcedureAddress
IProcedureAddress 724
IProcessId
IProcessId 727
IProfile
IProfile 730
IProfileHandle
IProfileHandle 741
IProgressIndicator
IProgressIndicator 748, 762
IProgressIndicatorNotifyHandler
IProgressIndicatorNotifyHandler 771

IPushButton
IPushButton 776, 780
IRadioButton
IRadioButton 787
IRecoordHandler
IRecoordHandler 797
IResizeEvent
IResizeEvent 800
IResizeHandler
IResizeHandler 803
IResource
IResource 805
IResourceId
IResourceId 808
IResourceLibrary
IResourceLibrary 812
IResourceLock
IResourceLock 820
is32Bit
IProcedureAddress 725
isAltDown
IKeyboardEvent 484
isAltKeyDown
IMouseEvent 629
isAnExtension
IFrameWindow 363
isAutoDeleteObject
IWindow 1067
isAutoDestroyWindow
IWindow 1068
isAutoScroll
IEntryField 274
isAutoSelect
I3StateCheckBox 9
ICheckBox 147
IRadioButton 787
ISettingButton 858
isAutoTab
IEntryField 274
isBidiSupported
IBidiSettings 126
isBitmap
IMenuItem 587
isCharacter
IKeyboardEvent 484
isChecked
IMenuDrawItemEvent 563
IMenuItem 585
isCommand
IEntryField 278
isComposite
IKeyboardEvent 484
isConversionNeeded
ICoordinateSystem 232
IScrollBar
IScrollBar 825
IScrollBarNotifyHandler
IScrollBarNotifyHandler 839
IScrollEvent
IScrollEvent 842
IScrollHandler
IScrollHandler 846
isCtrlDown
IKeyboardEvent 484
isCtrlKeyDown
IMouseEvent 629
isCursorSelect
IRadioButton 789
isDefault
IPushButton 777
isDisabled
IMenuDrawItemEvent 563
IMenuItem 585
isDragStarting
IEntryField 288
IMultiLineEdit 662
IWindow 1081
isDrawItem
IBaseListBox 93
IMenuItem 589
isDrawItemEnabled
IProgressIndicator 751
ISelectHandler
ISelectHandler 853
ISemaphoreHandle
ISemaphoreHandle 855
isEmpty
IBaseComboBox 57
IBaseListBox 85
IEntryField 285
isEnabled
IHandler 413
IWindow 1074
isEnabledForNotification
IWindow 1067
isEntryPoint32Bit
IDynamicLinkLibrary 264
ISettingButton
ISettingButton 858
ISettingButtonNotifyHandler

© IBM Corp. 1992, 1996

ISettingButtonNotifyHandler *(continued)*
ISettingButtonNotifyHandler 863
isExtendedSelect
IBaseListBox 93
isFastSpinEnabled
IBaseSpinButton 111
isFlashing
IFrameWindow 373
isForComposite
IKeyboardEvent 484
isFrame
IHelpSubitemNotFoundEvent 438
isFramed
IMenuItem 586
isFrameWindow
IFrameWindow 377
IMenu 537
IWindow 1050
isFromFrame
ICommandEvent 211
isGroup
IControl 223
IWindow 1050
isGUIInitialized
ICurrentThread 247
isHalftone
I3StateCheckBox 11
IStaticText 908
ISharedResource
ISharedResource 867
isHelp
IPushButton 778
isHide
ISWP 935
isHighlighted
IButton 136
IMenuItem 586
isHorizontal
IScrollBar 830
isHorizontalScroll
IBaseComboBox 63
IBaseListBox 93
IShowListHandler
IShowListHandler 872
isInsertMode
IEntryField 280
isInvalidComposite
IKeyboardEvent 484
isIPFCompatible
IHelpWindow 451
isItemChecked
IMenu 545
isItemEnabled
IMenu 545
isLayoutDistorted
IWindow 1065
ISlider
ISlider 878
ISliderArmHandler
ISliderArmHandler 890
ISliderDrawHandler
ISliderDrawHandler 893
isLike
IAcceleratorKey 24
isListShowing
IBaseComboBox 60
isMargin
IEntryField 281
isMaster
IBaseSpinButton 107
isMaximized
IFrameWindow 373
isMenu
IHelpSubitemNotFoundEvent 438
isMinimized
IFrameWindow 373
isMinimumSizeCachingEnabled
IWindow 1065
isModal
IFrameWindow 373
isMove
ISWP 935
isMultipleSelect
IBaseListBox 94
isNoAdjustPosition
IBaseListBox 94
isNoDismiss
IMenuItem 586
isOpen
IClipboard 162
IDynamicLinkLibrary 263
IResourceLibrary 813
ISpinHandler
ISpinHandler 898
isPMCompatible
IBaseSpinButton 107
IProgressIndicator 756
isPrimaryWindow
IWindow 1087

isRelatedHandle
IFrameWindow 377
IWindow 1080
isRepeat
IKeyboardEvent 484
isRibbonStripEnabled
IProgressIndicator 753
isScanCode
IKeyboardEvent 485
isSelectable
IMenuItem 589
isSelected
IBaseComboBox 62
IBaseListBox 90
IListBoxDrawItemEvent 510
IMenuDrawItemEvent 563
ISettingButton 859
isSelectionStateDrawn
IListBoxDrawItemEvent 510
isSeparator
IMenuItem 590
isServant
IBaseSpinButton 108
isSet
IAccelerator 18
isShiftDown
IKeyboardEvent 485
isShiftKeyDown
IMouseEvent 629
isShow
ISWP 935
isShowing
IWindow 1075
isSize
ISWP 935
isSnapToTickEnabled
IProgressIndicator 755
isSpinFieldValid
IBaseSpinButton 111
INumericSpinButton 675
ITextSpinButton 966
isStarted
IThread 996
ITimer 1017
isStrikeout
IStaticText 908
isSubmenu
IMenuItem 591
isSymmetricSwappingEnabled
IBidiSettings 123
isSystemCommand
IPushButton 779
isTabStop
IControl 223
IWindow 1050
IStaticText
IStaticText 903
isText
IMenuItem 587
isTopLevelShell
ICurrentThread 244
IStringHandle
IStringHandle 917
ISubmenu
ISubmenu 923
isUncombined
IKeyboardEvent 485
isUnderscore
IStaticText 908
isUndoable
IMultiLineEdit 661
isUpTransition
IKeyboardEvent 485
isValid
IAcceleratorTable::Cursor 40
IBaseComboBox::Cursor 77
IBaseListBox::Cursor 103
IClipboard::Cursor 168
IMenu 538
IMenu::Cursor 551
IProfile::Cursor 739
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
IThread::Cursor 1002
ITimer::Cursor 1020
ITitle 1030
IWindow 1050
IWindow::ChildCursor 1101
IWindowHandle 1109
isVertical
IProgressIndicator 745
IScrollBar 830
isVirtual
IKeyboardEvent 485
isVisible
IWindow 1075
isWindow
IHelpSubitemNotFoundEvent 438
isWindowValid
IWindow 1050

© IBM Corp. 1992, 1996

isWordByWordReorderingEnabled
IBidiSettings 123
isWordWrap
IMultiLineEdit 661
ISWP
ISWP 934
ISWPArray
ISWPArray 941
isWriteable
IBaseSpinButton 113
IEntryField 282
IMultiLineEdit 653
isXerrorCodeAvailable
ICurrentThread 244
ISystemBitmapHandle
ISystemBitmapHandle 942
ISystemMenu
ISystemMenu 947
ISystemPointerHandle
ISystemPointerHandle 952
isZOrder
ISWP 936
itemHandle
IBaseComboBox 58
IBaseListBox 86
itemHeight
IBaseListBox 87
itemHelpId
IMenu 538
itemId
IDrawItemEvent 259
itemIndex
IListBoxSizeItemEvent 522
itemPresSpaceHandle
IDrawItemEvent 259
itemProvider
IWindow 1059
itemRect
IDrawItemEvent 259
IMenu 538
itemSize
IListBoxSizeItemEvent 522
itemText
IBaseComboBox 57
IBaseListBox 85
ITextControl
ITextControl 958
ITextControlNotifyHandler
ITextControlNotifyHandler 960
ITextSpinButton
ITextSpinButton 964
ITextSpinButtonNotifyHandler
ITextSpinButtonNotifyHandler 978
IThread
IThread 982
IThreadFn
IThreadFn 1004
IThreadHandle
IThreadHandle 1006
IThreadId
IThreadId 1008
IThreadMemberFn
IThreadMemberFn 1011
ITimer
ITimer 1014
ITimerFn
ITimerFn 1021
ITimerMemberFn
ITimerMemberFn 1024
ITimerMemberFn0
ITimerMemberFn0 1027
ITitle
ITitle 1033
ITitleNotifyHandler
ITitleNotifyHandler 1041
IWindow
IWindow 1058, 1081
IWindowHandle
IWindowHandle 1107
IWindowNotifyHandler
IWindowNotifyHandler 1111

K

kAlt
IKey 467
kAltGraf
IKey 468
kBackSpace
IKey 468
kBackTab
IKey 468
kBoldId
ICommand 201
kBreak
IKey 468
kCancelId
ICommand 201

kCapsLock
IKey 468
kCloseId
ICommand 204
kCopyId
ICommand 201
kCopyToId
ICommand 201
kCtrl
IKey 468
kCutId
ICommand 201
kDelete
IKey 468
kDown
IKey 468
kEnd
IKey 469
kEnter
IKey 469
kEsc
IKey 469
IKey 466
IKeyboardHandler 493
IKey::KeyModifier 475
keyAt
IAcceleratorTable 37
IKeyboardConnectionTo 477
IKeyboardEvent 481
IKeyboardHandler 490
keyCount
IAcceleratorTable 37
keyModifier
IAcceleratorKey 27
keyName
ISharedResource 868
keysHelpId
IHelpHandler 425
kF1
IKey 469
kF10
IKey 469
kF11
IKey 469
kF12
IKey 469
kF13
IKey 469
kF14
IKey 469
kF15
IKey 470
kF16
IKey 470
kF17
IKey 470
kF18
IKey 470
kF19
IKey 470
kF2
IKey 470
kF20
IKey 470
kF21
IKey 470
kF22
IKey 471
kF23
IKey 471
kF24
IKey 471
kF3
IKey 471
kF4
IKey 471
kF5
IKey 471
kF6
IKey 471
kF7
IKey 471
kF8
IKey 471
kF9
IKey 472
kFileApplyId
ICommand 201
kFileCancelId
ICommand 201
kFileOkId
ICommand 202
kFontApplyId
ICommand 202
kFontCancelId
ICommand 202
kFontOkId
ICommand 202
kHelpId
ICommand 202

© IBM Corp. 1992, 1996

kHideId
ICommand 204
kHome
IKey 472
kInsert
IKey 472
kItalicId
ICommand 202
kLeft
IKey 472
kLocateId
ICommand 202
kMaximizeId
ICommand 204
kMinimizeId
ICommand 204
kMoveId
ICommand 204
kNewLine
IKey 472
kNoKey
IKey 472
kNumLock
IKey 472
kOkId
ICommand 203
kOpenId
ICommand 203
kPageDown
IKey 472
kPageUp
IKey 473
kPasteId
ICommand 203
kPause
IKey 473
kPrintId
ICommand 203
kRestoreId
ICommand 204
kRight
IKey 473
kSaveId
ICommand 203
kScrollLock
IKey 473
kSettingsId
ICommand 203
kShift
IKey 473
kSizeId
ICommand 204
kSpace
IKey 473
kSysRq
IKey 473
kTab
IKey 473
kUnderscoreId
ICommand 203
kUp
IKey 473
kWindowListId
ICommand 205

L

layoutAdjustment
IBaseComboBox 59
IWindow 1065
layoutType
IMenuItem 589
left
IStaticText 912
leftAlign
IBaseSpinButton 117
IEntryField 293
leftIndex
IEntryField 280
lib files, description xxiii
libibmui.a xxv
libibmuis.a xxv
limit
IBaseComboBox 64
IBaseSpinButton 111
IEntryField 286
IMultiLineEdit 659
IStaticText 905
limitId
IEntryField 291
IMultiLineEdit 664
IStaticText 911
lineDown
IScrollHandler 847
lineLeft
IScrollHandler 847
lineRight
IScrollHandler 848
lineUp
IScrollHandler 848

IListBox 495
IListBox::Style 508
IListBoxDrawItemEvent 509
IListBoxDrawItemHandler 513
IListBoxNotifyHandler 518
IListBoxSizeItemEvent 521
listShown
    IShowListHandler 872
loadAccelTable
    IResourceLibrary 813
loadBitmap
    IResourceLibrary 814
loadDialog
    IResourceLibrary 814
loadHelpTable
    IResourceLibrary 815
loadIcon
    IResourceLibrary 815
loadMenu
    IResourceLibrary 816
loadMessage
    IResourceLibrary 817
loadPointer
    IResourceLibrary 817
loadString
    IResourceLibrary 817
locateKeyLike
    IAcceleratorTable 37
locateText
    IBaseComboBox 61
    IBaseListBox 89
location
    IFrameExtension 332
lock
    IResource 806
lostFocus
    IFocusHandler 324
lowerCase
    IEntryField 293
ILowEventParameter 524
lowHighByte
    IEventData 316
lowLowByte
    IEventData 316
lowNumber
    IEventData 316

M

makePopUpMenu
    IMenuHandler 578
mapPoint
    IWindow 1078
margin
    IEntryField 293
master
    IBaseSpinButton 117
matchForMnemonic
    IFrameWindow 365
    IWindow 1065
maximize
    IFrameWindow 371
maximizeButton
    IFrameWindow 386
maximized
    IFrameWindow 386
maximizeRect
    IFrameWindow 371
member function names, conventions xxv
IMenu 525
IMenu::Cursor 550
IMenu::Style 553
menuBar 554
    IFrameWindow 386
IMenuBar::Style 561
menuBarCommand
    IHelpHandler 425
IMenuDrawItemEvent 562
IMenuDrawItemHandler 565
IMenuDrawItemHandler::DrawFlag 570
menuEnded
    IMenuHandler 579
IMenuEvent 571
menuHandle 574
    IMenu 538
    ISubmenu 925
IMenuHandler 576
menuItem 582
    IMenu 539
    IMenuEvent 572
IMenuItem::Attribute 599
IMenuItem::Style 600
menuItemId
    IMenuEvent 572
IMenuNotifyHandler 601
menuSelected
    IMenuHandler 579

© IBM Corp. 1992, 1996

menuShowing
IMenuHandler 579
IMessageBox 604
IMessageBox::Style 613
messageQueue
ICurrentThread 248
IThread 989
IWindow 1051
IMessageQueueHandle 615
metafileFormat
IClipboard 165
minimize
IFrameWindow 371
minimizeButton
IFrameWindow 386
minimized
IFrameWindow 387
minimizedIcon
IFrameWindow 387
minimizeRect
IFrameWindow 372
minimumCharacters
IBaseListBox 87
minimumRows
IBaseComboBox 59
IBaseListBox 87
minimumSize
IWindow 1065
minScrollIncrement
IScrollBar 830
mixedCharacter
IKeyboardEvent 482
mixedData
IBaseComboBox 71
IEntryField 293
mnemonic
IStaticText 913
IModuleHandle 618
mouseAction
IMouseClickEvent 621
mouseButton
IMouseClickEvent 621
mouseClicked
IMouseConnectionTo 626
IMouseHandler 633
IMouseClickEvent 620
IMouseConnectionTo 624
IMouseEvent 628
IMouseHandler 631
mouseMoved
IMouseConnectionTo 626
IMouseHandler 633
mousePointer
IFrameWindow 367
mousePointerChange
IMouseConnectionTo 626
IMouseHandler 633
IMousePointerHandler 640
IMousePointerEvent 635
IMousePointerHandler 638
mousePosition
IMenuEvent 572
IMouseEvent 629
moveable
IMessageBox 609
moveArmToPixel
IProgressIndicator 746
moveArmToTick
IProgressIndicator 746
movePointerTo
IWindow 1072
moveScrollBox
IScrollHandler 849
moveScrollBoxTo
IScrollBar 827
moveSizeTo
IBaseComboBox 65
IBaseSpinButton 114
IEntryField 287
IGroupBox 400
IHelpWindow 454
IOutlineBox 693
IProgressIndicator 761
IScrollBar 833
IStaticText 908
IWindow 1078
moveSizeToClient
IFrameWindow 368
moveTo
IHelpWindow 455
IWindow 1078
moving
ISliderArmHandler 890
IMultiLineEdit 641
IMultiLineEdit::Style 668
IMultiLineEditNotifyHandler 669
Multimedia Classes xxii
overview xxii

multipleSelect
IBaseListBox 99

N

name
IProfile 732
nativeOrientation
ICoordinateSystem 232
nativeRect
IBaseComboBox 65
ITitle 1037
IWindow 1078
newScrollBoxPosition
IScrollEvent 843
newSize
IResizeEvent 801
newStartedThread
IThread 999
nextShellRect
IFrameWindow 372
noAdjustPosition
IBaseListBox 99
noAttribute
IMenuItem 594
noBorder
IBaseSpinButton 118
IPushButton 782
noCursorSelect
IRadioButton 793
noDismiss
IMenuItem 594
noHandle
IThreadHandle 1007
noIcon
IMessageBox 610
noModifier
IKey 467
noMoveWithOwner
IFrameWindow 387
noPointerFocus
IButton 140
noSelections
IRadioButton 793
noStyle
IHelpWindow 457
IMenu 548
IMenuItem 595
IWindow 1097
notFound
IBaseComboBox 69
IBaseListBox 97
notificationHandler
IWindow 1091
notifyObservers
IWindow 1068, 1092
notifyOwner
IFrameWindow 366
number1
IEventData 316
number2
IEventData 316
numberOfApplications
IProfile 732
numberOfExtensions
IFrameHandler 346
numberOfItems
IMenu 539
numberOfKeys
IProfile 732
numberOfLines
IMultiLineEdit 651
numberOfSelections
IBaseComboBox 62
IBaseListBox 90
numberOfTicks
IProgressIndicator 757
numeralDisplay
IBidiSettings 124
numerations, conventions xxv
INumericSpinButton 672
INumericSpinButton::Style 682
INumericSpinButtonNotifyHandler 683

O

objectText
ITitle 1035
objectTextId
ITitle 1039
IObjectWindow 686
IWindow 1051
observerList
IWindow 1092
oemData
IBaseComboBox 71
okButton
IMessageBox 610

© IBM Corp. 1992, 1996

okCancelButton
IMessageBox 610
oldSize
IResizeEvent 801
open
IClipboard 162
IDynamicLinkLibrary 264
openLibrary
IHelpHandler 425
operator _WidgetRec *
IWindowHandle 1108
operator _XDisplay *
IDisplayHandle 256
operator _XGC *
IPresSpaceHandle 718
operator _XrmHashBucketRec *
IProfileHandle 742
operator _XtAppStruct *
IContextHandle 219
operator !=
IAcceleratorKey 24
IColor 176
operator []
ISWPArray 940
operator =
IAcceleratorKey 27
IAcceleratorTable 36
IBitmapHandle 132
IColor 177
IDynamicLinkLibrary 262
IEvent 307
IFrameEvent 327
IMenuItem 585
IPointerHandle 710
IProfile 730
IResourceLibrary 812
ISWP 935
IThread 998
ITimer 1015
operator ==
IAcceleratorKey 24
IColor 176
ITimer 1013
operator char *
IEventData 317
operator PtrToFnType
IProcedureAddress 725
operator unsigned long
IEventData 317
IResourceId 810
operator Value
IAccelTblHandle 43
IAnchorBlockHandle 45
IBitmapHandle 132
IEnumHandle 302
IHandle 409
IMessageQueueHandle 616
IModuleHandle 618
IPointerHandle 710
IProcessId 727
ISemaphoreHandle 855
IStringHandle 917
IThreadId 1008
IOutlineBox 689
IOutlineBox::Style 697
outlineType
IOutlineBox 691
owner
IAccelerator 21
IClipboard 161
IMenu 538
ITitle 1030
IWindow 1069
ownerItemData
IDrawItemEvent 260

P

padWithZeros
INumericSpinButton 680
pageDown
IScrollHandler 848
pageLeft
IScrollHandler 848
pageRight
IScrollHandler 848
pageScrollIncrement
IScrollBar 830
pageUp
IScrollHandler 848
IPaintConnectionTo 698
IPaintEvent 701
IPaintHandler 706
paintWindow
IPaintConnectionTo 700
IPaintHandler 708
paletteFormat
IClipboard 165
parameter1
IEvent 308

parameter2
IEvent 308
parent
IWindow 1069
parentSize
ITitle 1037
IWindow 1079
passEventToOwner
IBaseListBox 95
IEntryField 288
IMultiLineEdit 663
IPushButton 780
IScrollBar 834
ISettingButton 860
IStaticText 909
IWindow 1083
passToOwner
IEvent 309
paste
IEntryField 277
IMultiLineEdit 646
pib
ICurrentApplication 241
pmCompatible
IBaseSpinButton 118
IProgressIndicator 766
IPointerHandle 709
pointerPosition
IWindow 1072
IPopUpMenu 712
position
IBaseComboBox 65
IGroupBox 400
IOutlineBox 693
IScrollBar 833
ISWP 936
IWindow 1079
positionBehindSibling
IWindow 1073
positionBehindSiblings
IWindow 1073
positionChanged
ISliderArmHandler 890
positionExtensions
IFrameHandler 348
positionId
IWindow 1095
positionOnSiblings
IWindow 1073
postEvent
postEvent *(continued)*
IMessageQueueHandle 616
IWindow 1059
IWindowHandle 1108
postEvents
IMessageQueueHandle 616
IWindowHandle 1108
postHelp
IMenuItem 595
postSystemCommand
IMenuItem 596
prepareForUse
IWindow 1087
Presentation Manager messages
WM_CONTEXTMENU 576
WM_INITMENU 572, 576
WM_MENUEND 572, 576
WM_MENUSELECT 572, 576
presSpace
IWindow 1076
presSpaceHandle 717
IPaintEvent 704
prevScrollBoxPosition
IScrollBar 827
primaryFormat
IClipboard 160
primaryScale
IProgressIndicator 753
primaryScale1
IProgressIndicator 766
primaryScale2
IProgressIndicator 766
priorityClass
IThread 997
priorityLevel
IThread 997
IPrivateResource 719
procAddress
IDynamicLinkLibrary 265
IProcedureAddress 723
IProcessId 727
processMsgs
ICurrentThread 248
IProfile 729
IProfile::Cursor 738
IProfileHandle 741
IProgressIndicator 743
IProgressIndicator::Style 770
IProgressIndicatorNotifyHandler 771
IPushButton 774

© IBM Corp. 1992, 1996

IPushButton::Style 785

Q

queryIcon
IMessageBox 610
queueSize
IThread 989

R

IRadioButton 786
IRadioButton::Style 795
range
INumericSpinButton 675
readOnly
IBaseSpinButton 118
IEntryField 293
IMultiLineEdit 665
readOnlyDropDownType
IBaseComboBox 71
IRecoordHandler 796
rect
IGroupBox 400
IPaintEvent 704
IWindow 1079
redMix
IColor 175
refresh
IWindow 1076
registerCallbacks
IBaseComboBox 67
IBaseListBox 95
IBaseSpinButton 115
ICheckBox 150
IEntryField 289
IFrameWindow 378
IMenuBar 558
IMultiLineEdit 663
INumericSpinButton 678
IPopUpMenu 715
IProgressIndicator 762
IPushButton 781
IRadioButton 791
IScrollBar 834
IWindow 1083
registerFormat
IClipboard 161
registerFrameClass
IFrameWindow 382
relatedHandlesList
IThread 988
relativeSize
IFrameExtension 333
releasePointer
IWindow 1072
releasePresSpace
IWindow 1077
remainingStack
ICurrentThread 244
remove
IAccelerator 18
IComboBox 189
IListBox 502
removeAll
IComboBox 189
IEntryField 286
IListBox 502
IMultiLineEdit 660
ITextSpinButton 969
removeAllKeys
IAcceleratorTable 32
removeAllObservers
IWindow 1092
removeAt
IComboBox 189
IListBox 502
ITextSpinButton 970
removeBorder
IBaseSpinButton 108
IPushButton 775
removeConditionalCascade
IMenu 539
removeDefaultHandler
IFrameWindow 382
removeDetent
ISlider 881
removeExtension
IFrameWindow 363
removeFromWindowList
IFrameWindow 376
removeFromWindowSet
IWindow 1084
removeHandler
IWindow 1088
removeId
IComboBox 193
IListBox 505
removeKeyAt
IAcceleratorTable 32

removeKeyLike
IAcceleratorTable 33
removeLine
IMultiLineEdit 651
removeObserver
IWindow 1092
removeRelatedHandleFromWindowSet
IWindow 1084
removeSourceEmphasis
IMenuHandler 580
removeSubmenu
IMenu 541
ISubmenu 925
removeSubmenuAt
IMenu 537
ISubmenu 924
renderAllFormats
IClipboardHandler 172
renderFormat
IClipboardHandler 172
repeatCount
IKeyboardEvent 482
replaceAt
IComboBox 189
IListBox 502
ITextSpinButton 970
replaceKeyAt
IAcceleratorTable 33
reset
IAccelerator 18
resetActiveColor
IWindow 1054
resetActiveTextBackgroundColor
ITitle 1032
resetActiveTextForegroundColor
ITitle 1032
resetBackgroundColor
IBaseSpinButton 109
IFrameWindow 356
IMenu 533
IWindow 1054
resetBorderColor
IWindow 1054
resetColor
IWindow 1088
resetDisabledBackgroundColor
IFrameWindow 356
IMenu 534
IWindow 1054
resetDisabledForegroundColor
resetDisabledForegroundColor *(continued)*
IMenu 534
IWindow 1054
resetFillColor
IStaticText 902
resetFont
IWindow 1063
resetForegroundColor
IBaseSpinButton 109
IMenu 534
IScrollBar 824
IWindow 1054
resetHiliteBackgroundColor
IMenu 534
IWindow 1054
resetHiliteForegroundColor
IMenu 534
IScrollBar 824
IWindow 1055
resetInactiveColor
IWindow 1055
resetInactiveTextBackgroundColor
ITitle 1032
resetInactiveTextForegroundColor
ITitle 1032
resetMinimumSize
IWindow 1066
resetShadowColor
IWindow 1055
resetTextChangedFlag
IEntryField 285
IMultiLineEdit 643
IResizeEvent 800
IResizeHandler 802
IResource 805
IResourceId 808
resourceLibrary 811
ICurrentApplication 238
IResourceId 810
IResourceLock 820
restore
IFrameWindow 372
restoreRect
IFrameWindow 372
result
IEvent 309
IFrameWindow 366
resume
IThread 992
retryCancelButton

© IBM Corp. 1992, 1996

retryCancelButton *(continued)*
IMessageBox 610
ribbonStrip
IProgressIndicator 767
right
IStaticText 913
rightAlign
IBaseSpinButton 118
IEntryField 293
run
ICurrentApplication 240
IThreadFn 1005
IThreadMemberFn 1011

S

sample directory location xxviii
saveBits
IWindow 1097
saved
IFrameHandler 346
savedMinimumSize
IWindow 1091
saveMinimumSize
IWindow 1091
sbcsData
IBaseComboBox 71
IEntryField 293
scaleId
IProgressIndicator 764
scanCode
IKeyboardEvent 483
scanCodeKeyPress
IKeyboardHandler 493
scrollableRange
IScrollBar 828
scrollAmount
IScrollEvent 843
IScrollBar 823
IScrollBar::Style 838
IScrollBarNotifyHandler 839
scrollBarWindow
IScrollEvent 843
scrollBoxPosition
IScrollBar 828
scrollBoxPositionId
IScrollBar 835
scrollBoxRange
IScrollBar 828
scrollBoxTrack
IScrollHandler 849
scrollBoxTrackEnd
IScrollHandler 849
scrollEnd
IScrollHandler 849
IScrollEvent 842
IScrollHandler 845
searchListWindow
IHelpWindow 454
select
IBaseComboBox 62
IBaseListBox 90
ISettingButton 859
selectAll
IBaseListBox 91
selected
ISelectHandler 854
selectedIndex
IRadioButton 789
selectedRange
IEntryField 283
IMultiLineEdit 654
selectedText
IEntryField 283
IMultiLineEdit 654
selectedTextLength
IEntryField 283
IMultiLineEdit 654
selectHalftone
I3StateCheckBox 11
ISelectHandler 851
selectId
IBaseComboBox 69
IBaseListBox 97
ISettingButton 861
selection
IBaseComboBox 63
IBaseListBox 91
selectItem
IListBoxDrawItemHandler 516
IMenu 545
selectRange
IEntryField 283
IMultiLineEdit 655
ISemaphoreHandle 855
sendEvent
IHelpWindow 451
IWindow 1060
IWindowHandle 1109

sendEvents
IWindowHandle 1109
separator
IMenuItem 596
separatorHandle
IFrameExtension 336
separatorType
IFrameExtension 336
separatorWidth
IFrameExtension 336
servant
IBaseSpinButton 118
set
IAccelerator 19
setAccelerator
IHelpWindow::Settings 461
setAcceleratorHandle
IWindow 1048
setAcceleratorTable
IWindow 1048
setAccelResLibrary
IHelpWindow::Settings 462
setActiveColor
IWindow 1055
setActiveTextBackgroundColor
ITitle 1033
setActiveTextForegroundColor
ITitle 1033
setActiveWindow
IHelpWindow 444
setAlignment
IBaseSpinButton 108
IEntryField 273
IStaticText 901
setApplicationOrientation
ICoordinateSystem 232
setArgs
ICurrentApplication 238
setArmSize
ISlider 877
setAssociatedWindow
IHelpWindow 444
setAttribute
IMenuItem 592
setAutoDeleteObject
IWindow 1068
setAutoDestroyWindow
IWindow 1068
setAutoInitGUI
IThread 987
setBackgroundColor
IBaseComboBox 54
IBaseSpinButton 109
IMenu 534
IProgressIndicator 747
IWindow 1055
setBitmap
IClipboard 157
IMenu 541
IMenuItem 587
ISubmenu 926
setBlue
IColor 175
setBorderColor
IWindow 1056
setBorderHeight
IFrameWindow 353
setBorderSize
IFrameWindow 354
setBorderWidth
IFrameWindow 355
setCharType
IEntryField 273
setChecked
IMenuItem 586
setClient
IFrameWindow 355
setClientRect
IFrameFormatEvent 340
setColor
IGUIColor 406
IWindow 1088
setCommand
IAcceleratorKey 23
IMenuItem 584
setConditionalCascade
IMenu 539
setControlHandle
IEvent 311
setCursorLinePosition
IMultiLineEdit 651
setCursorPosition
IEntryField 283
IMultiLineEdit 652
setData
IClipboard 158
setDefaultApplicationName
IProfile 731
setDefaultAttribute
IMenuItem 584

© IBM Corp. 1992, 1996

setDefaultAutoInitGUI
IThread 987
setDefaultOrdering
IFrameWindow 369
IWindow 1073
setDefaultQueueSize
IThread 990
setDefaultStackSize
IThread 991
setDefaultStyle
I3StateCheckBox 12
IBaseComboBox 63
IBaseListBox 94
ICheckBox 149
IComboBox 190
IEntryField 284
IFrameWindow 370
IGroupBox 399
IHelpWindow 451
IListBox 503
IMenu 546
IMenuBar 556
IMenuItem 591
IMultiLineEdit 657
INumericSpinButton 677
IOutlineBox 692
IProgressIndicator 756
IPushButton 778
IRadioButton 790
IScrollBar 832
ISlider 881
IStaticText 906
ITextSpinButton 968
setDestroyOnClose
IFrameWindow 374
setDisabled
IMenuItem 586
setDisabledBackgroundColor
IMenu 534
IWindow 1056
setDisabledForegroundColor
IMenu 535
IWindow 1056
setDispatchingHandle
IEvent 311
setDrawItem
IMenuItem 590
setEditRegion
IMultiLineEdit 655
setEditRegionHeight
IMultiLineEdit 656
setEditRegionWidth
IMultiLineEdit 656
setEventType
IEvent 309
setExceptionFunction
IWindow 1062
setExtendedStyle
IMenuItem 592
IWindow 1088
setExtensions
IFrameWindow 379
setExtensionSize
IFrameWindow 363
setFillColor
IStaticText 903
setFocus
IWindow 1051
setFont
IMultiLineEdit 648
IWindow 1063
setForegroundColor
IBaseComboBox 55
IBaseSpinButton 110
IMenu 535
IOutlineBox 690
IProgressIndicator 748
IScrollBar 825
IWindow 1056
setFramed
IMenuItem 586
setGraphicContext
IPaintEvent 704
setGreen
IColor 175
setHandle
IClipboard 158
IEvent 312
setHelpId
IMenuItem 593
IWindow 1064
setHelpKey
IAcceleratorKey 23
setHelpResLibrary
IHelpWindow::Settings 460
setHelpTable
IHelpWindow 449
IHelpWindow::Settings 460

setHide
ISWP 936
setHighlighted
IMenuItem 587
setHiliteBackgroundColor
IMenu 535
IWindow 1056
setHiliteForegroundColor
IMenu 535
IScrollBar 825
IWindow 1056
setHomePosition
IProgressIndicator 752
setIcon
IFrameWindow 352
setId
IApplication 50
IWindow 1051
setInactiveColor
IWindow 1057
setInactiveTextBackgroundColor
ITitle 1033
setInactiveTextForegroundColor
ITitle 1033
setIndex
IMenuItem 589
setInterval
ITimer 1016
setItem
IMenu 542
ISubmenu 927
setItemHandle
IBaseComboBox 58
IBaseListBox 86
setItemHeight
IBaseListBox 87
setItemHelpId
IMenu 540
setItemProvider
IWindow 1059
setItemSize
IListBoxDrawItemHandler 517
IListBoxSizeItemEvent 522
setItemText
IBaseComboBox 57
IBaseListBox 86
setKey
IAcceleratorKey 28
setLayout
IMenuItem 589
setLayoutDistorted
IBaseComboBox 68
IBaseListBox 87
IEntryField 290
IFrameWindow 375
ITextControl 956
IWindow 1066
setLeftIndex
IEntryField 280
setLibraries
IHelpWindow::Settings 460
setLimit
IBaseComboBox 64
IBaseSpinButton 111
IEntryField 286
IMultiLineEdit 659
INumericSpinButton 674
IStaticText 905
ITextSpinButton 966
setLock
IResourceLock 822
setMaster
IBaseSpinButton 112
setMenu
IMenuBar 556
setMenuBar
IHelpWindow::Settings 462
setMinimumCharacters
IBaseListBox 88
setMinimumRows
IBaseComboBox 59
IBaseListBox 88
setMinimumSize
IWindow 1066
setMinScrollIncrement
IScrollBar 830
setMousePointer
IFrameWindow 367
IMousePointerEvent 636
setMove
ISWP 936
setNoAdjust
ISWP 936
setNoDismiss
IMenuItem 587
setNotificationHandler
IWindow 1092
setNumeralDisplay
IBidiSettings 124
setObjectText

© IBM Corp. 1992, 1996

setObjectText *(continued)*
ITitle 1035
setOutlineType
IOutlineBox 692
setOwner
IClipboard 164
IWindow 1070
setPageScrollIncrement
IScrollBar 831
setParent
IWindow 1070
setPassToOwner
IEvent 309
setPosition
ISWP 937
setPrevScrollBoxPosition
IScrollBar 828
setPrimaryScale
IProgressIndicator 753
setPriority
IApplication 48
IThread 998
setQueueSize
IThread 990
setRange
INumericSpinButton 676
setRed
IColor 175
setRelatedHandlesList
IThread 988
setResourceLibrary
ICurrentApplication 239
setRestoreRect
IFrameWindow 373
setResult
IEvent 309
IFrameWindow 366
setScrollableRange
IScrollBar 829
setScrollBar
IScrollBar 829
setSelectable
IMenuItem 590
setSelectionStateDrawn
IListBoxDrawItemEvent 510
setSeparator
IMenuItem 590
setSeparatorHandle
IFrameExtension 336
setShadowColor
IWindow 1057
setShaftBreadth
IProgressIndicator 754
setShaftPosition
IProgressIndicator 754
setShow
ISWP 937
setSize
IFrameExtension 333
IMenuDrawItemHandler 568
ISWP 937
setSizeFlag
ISWP 937
setStackSize
IThread 991
setStyle
IEntryField 289
IMenuItem 593
IWindow 1089
setSubmenu
IMenu 542
setSubmenuHandle
IMenuItem 591
setSystemCommand
IAcceleratorKey 24
setTab
IMultiLineEdit 657
setText
IButton 138
IClipboard 159
IGroupBox 398
IMenu 543
IMenuItem 588
IMultiLineEdit 659
IStaticText 908
ISubmenu 927
ITextControl 956
ITextSpinButton 967
ITitle 1034
setTextChangedFlag
IEntryField 285
IMultiLineEdit 644
setTextOrientation
IBidiSettings 124
setTextShape
IBidiSettings 124
setTextType
IBidiSettings 124

setTickLength
IProgressIndicator 757
setTicks
IProgressIndicator 758
setTickText
IProgressIndicator 759
ISettingButton 857
ISettingButtonNotifyHandler 863
Settings
IHelpWindow::Settings 459
setTitle
IHelpWindow 452
IHelpWindow::Settings 462
IMessageBox 607
setTitleText
ITitle 1036
setToFirst
IAcceleratorTable::Cursor 40
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 102
IClipboard::Cursor 168
IMenu::Cursor 551
IProfile::Cursor 739
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
IThread::Cursor 1002
ITimer::Cursor 1020
IWindow::ChildCursor 1101
setToIndex
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 102
setToLast
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 102
IMenu::Cursor 551
IProfile::Cursor 740
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
setToNext
IAcceleratorTable::Cursor 40
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 103
IClipboard::Cursor 168
IMenu::Cursor 551
IProfile::Cursor 740
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
IThread::Cursor 1002
ITimer::Cursor 1020
IWindow::ChildCursor 1101
setToolBarList
IFrameWindow 374
setTop
IBaseComboBox 60
IBaseListBox 88
IMultiLineEdit 652
setTopLevelShell
ICurrentThread 249
setToPrevious
IBaseComboBox::Cursor 76
IBaseListBox::Cursor 103
IMenu::Cursor 551
IProfile::Cursor 740
ISubmenu::Cursor 932
ITextSpinButton::Cursor 975
setTutorial
IHelpWindow::Settings 461
setUserResourceLibrary
ICurrentApplication 239
setUsingHelp
IHelpWindow 449
IHelpWindow::Settings 461
setValue
INumericSpinButton 676
setVariable
IThread 996
setViewNumber
ITitle 1036
setViewText
ITitle 1036
setVirtualKeyMemberFn
IKeyboardConnectionTo 478
setVisibleCount
IScrollBar 829
setWindowData
IWindow 1089
setWindowLayout
IBidiSettings 124
setWindowList
IThread 988
setXerrorCode
ICurrentThread 249
setZOrder
ISWP 937
shadowColor
IWindow 1057
shadowColorId
IWindow 1095
shaftPosition
IProgressIndicator 754

© IBM Corp. 1992, 1996

shaftSize
IProgressIndicator 755
ISharedResource 866
shareParentDBCSStatus
IFrameWindow 361
shellPosition
IFrameWindow 387
shift
IKey 467
show
IBaseListBox 94
IFrameWindow 374
IGroupBox 400
IHelpWindow 449
IMessageBox 605
IPopUpMenu 714
IWindow 1077
showContents
IHelpHandler 425
showCoverPage
IHelpHandler 426
showHistory
IHelpHandler 426
showIndex
IHelpHandler 426
showList
IBaseComboBox 60
IShowListHandler 870
showModally
IFrameWindow 367
showPage
IHelpHandler 426
showPanelIds
IHelpWindow 450
showSearchList
IHelpHandler 426
showSourceEmphasis
IWindow 1077
showTutorial
IHelpHandler 427
simpleType
IBaseComboBox 72
size
IBaseComboBox 65
IGroupBox 400
IOutlineBox 693
IScrollBar 833
ISWP 937
ISWPArray 941
IWindow 1079
sizeBitmapTo
IResourceLibrary 817
sizeId
IWindow 1096
sizeTo
IFrameExtension 334
IHelpWindow 455
IWindow 1079
sizingBorder
IFrameWindow 387
sleep
ICurrentThread 245
ISlider 874
ISlider::Style 887
ISliderArmHandler 888
ISliderDrawHandler 892
snapToTickMark
IProgressIndicator 767
source
ICommandEvent 211
spinDown
IBaseSpinButton 112
INumericSpinButton 675
ITextSpinButton 966
spinEnded
ISpinHandler 899
ISpinHandler 896
spinTo
INumericSpinButton 676
ITextSpinButton 970
spinUp
IBaseSpinButton 112
INumericSpinButton 675
ITextSpinButton 966
split
IMenuItem 596
splitWithSeparator
IMenuItem 596
stackSize
IThread 992
Standard Control Classes xxii
overview xxii
start
IFrameWindow 365
IThread 992
ITimer 1017
startedThread
ICurrentThread 251
IThread 999

startHandlingEventsFor
IWindow 1084
IStaticText 900
IStaticText::Style 916
stop
IThread 994
ITimer 1017
stopHandlingEventsFor
IHandler 414
IHelpHandler 423
stopProcessingMsgs
IThread 987
strikeout
IStaticText 913
strikeoutId
IStaticText 911
IStringHandle 917
style
IMenuItem 593
IWindow 1090
subitemNotFound
IHelpHandler 427
ISubmenu 919
ISubmenu::Cursor 931
submenuHandle
IMenuItem 591
subtopicId
IHelpSubitemNotFoundEvent 438
suspend
ICurrentThread 250
IThread 995
swapPage
IHelpHandler 427
ISWP 934
swpArray 940
IFrameFormatEvent 340
synchPaint
IWindow 1097
ISystemBitmapHandle 942
systemColor
IColor 175
IGUIColor 406
systemCommand
ICommandConnectionTo 208
ICommandHandler 217
IPushButton 783
systemCommandId
IWindow 1096
systemMenu 946
IFrameWindow 387
systemModal
IFrameWindow 388
IMessageBox 610
ISystemPointerHandle 952
systemProfile
IProfile 736
systemScrollBarWidth
IScrollBar 832
systemScrollBoxLength
IScrollBar 832
systemScrollButtonLength
IScrollBar 832

T

tabStop
IControl 224
terminateGUI
ICurrentThread 248
text
IClipboard 159
IEntryField 282
IMenuItem 588
IMultiLineEdit 660
ITextControl 957
ITextSpinButton 967
ITitle 1035
text widgets 272, 642
ITextControl 955
ITextControlNotifyHandler 960
textFormat
IClipboard 166
textId
IBaseSpinButton 116
ITextControl 959
textLength
IMultiLineEdit 660
ITextControl 957
ITitle 1035
textOrientation
IBidiSettings 125
textShape
IBidiSettings 125
ITextSpinButton 963
ITextSpinButton::Cursor 974
ITextSpinButton::Style 977
ITextSpinButtonNotifyHandler 978
textType
IBidiSettings 125

© IBM Corp. 1992, 1996

IThread 981
IThread::Cursor 1001
IThreadFn 1004
IThreadHandle 1006
threadId 1008
IThread::Cursor 1002
IThreadMemberFn 1010
tickLength
IProgressIndicator 759
tickPosition
IProgressIndicator 760
tickSpacing
IProgressIndicator 760
tickText
IProgressIndicator 760
ITimer 1013
ITimer::Cursor 1019
timerAt
ITimer 1016
timerExpired
ITimerFn 1022
ITimerMemberFn 1024
ITimerMemberFn0 1027
ITimerFn 1021
ITimerMemberFn 1023
ITimerMemberFn0 1026
ITitle 1029
titleBar
IFrameWindow 388
ITitleNotifyHandler 1041
toolBarList
IFrameWindow 374
top
IBaseComboBox 60
IBaseListBox 89
IMultiLineEdit 652
IStaticText 913
topHandle
IBaseComboBox 55
IBaseListBox 84
IBaseSpinButton 110
IFrameWindow 357
IMultiLineEdit 647
IPopUpMenu 713
IWindow 1057
topicId
IHelpSubitemNotFoundEvent 438
totalRectFor
IFrameExtension 334
tryToLoadBitmap
IResourceLibrary 817
tryToLoadDialog
IFrameWindow 382
tryToLoadIcon
IResourceLibrary 818
tryToLoadMessage
IResourceLibrary 819
tryToLoadString
IResourceLibrary 819
tutorialName
IHelpTutorialEvent 441
Two-Dimensional Graphic Classes xxii
overview xxii
type
IBaseComboBox 64
IFrameExtension 335
type names, conventions xxv

U

ulAltFlag
IKeyboardEvent 486
ulAltMask
IKeyboardEvent 486
ulCharacterFlag
IKeyboardEvent 487
ulCompositeFlag
IKeyboardEvent 487
ulCtrlFlag
IKeyboardEvent 487
ulCtrlMask
IKeyboardEvent 486
ulForCompositeFlag
IKeyboardEvent 487
ulInvalidCompositeFlag
IKeyboardEvent 487
ulRepeatFlag
IKeyboardEvent 487
ulReserved1
ISWP 939
ulReserved2
ISWP 939
ulScanCodeFlag
IKeyboardEvent 487
ulShiftFlag
IKeyboardEvent 487
ulShiftMask
IKeyboardEvent 486

ulUncombinedFlag
IKeyboardEvent 488
ulUpTransitionFlag
IKeyboardEvent 488
ulVirtualFlag
IKeyboardEvent 488
unavailable
IMenuItem 597
uncheckItem
IMenu 546
ISubmenu 928
underscore
IStaticText 913
underscoreId
IStaticText 911
undo
IMultiLineEdit 661
ISubmenu 928
unhighlight
IButton 136
IMenuDrawItemHandler 568
uniqueKeyFor
IAcceleratorKey 29
unlock
IResource 806
unreadable
IEntryField 294
unregisterCallbacks
IBaseComboBox 67
IBaseListBox 96
IBaseSpinButton 115
ICheckBox 151
IEntryField 289
IFrameWindow 378
IMenuBar 558
IMultiLineEdit 663
INumericSpinButton 679
IPopUpMenu 715
IProgressIndicator 762
IPushButton 781
IRadioButton 792
IScrollBar 834
IWindow 1085
update
IFrameWindow 364
upperCase
IEntryField 294
useExtensionMinimumSize
IFrameWindow 364
useMinimumSize
useMinimumSize *(continued)*
IFrameExtension 334
userProfile
IProfile 736
userResourceLibrary
ICurrentApplication 240
usesDialogBackground
IFrameWindow 356

V

value
IColor 175
INumericSpinButton 676
valueId
IBaseSpinButton 117
variable
IThread 996
vertCenter
IStaticText 913
vertical
IProgressIndicator 767
IScrollBar 836
verticalScroll
IFrameWindow 388
IMultiLineEdit 665
viewedPagesWindow
IHelpWindow 454
viewNumber
ITitle 1036
viewNumberId
ITitle 1039
viewText
ITitle 1037
viewTextId
ITitle 1039
virtualKey
IAcceleratorKey 28
IKeyboardEvent 483
virtualKeyPress
IKeyboardConnectionTo 479
IKeyboardHandler 493
visible
IWindow 1097
visibleCount
IScrollBar 829
visibleId
IWindow 1096
visibleLines
IMultiLineEdit 652

© IBM Corp. 1992, 1996

visibleRectangle
IBaseComboBox 59
IGroupBox 398
IOutlineBox 691
IWindow 1067

W

waitFor
ICurrentThread 245
waitForAllThreads
ICurrentThread 245
waitForAnyThread
ICurrentThread 246
warningIcon
IMessageBox 610
willDestroyOnClose
IFrameWindow 374
window 1044
IEvent 312
IWindow::ChildCursor 1100
IWindow::ExceptionFn 1103
IWindow::Style 1105
windowHandle 1107
IHelpNotifyEvent 435
ISWP 937
windowId
IMousePointerEvent 636
windowLayout
IBidiSettings 125
windowList
IFrameWindow 388
IThread 988
IWindowNotifyHandler 1111
windowResize
IResizeHandler 804
windowSizeChanged
IRecoordHandler 798
windowULong
IWindow 1090
windowUnderPointer
IMouseEvent 630
windowUShort
IWindow 1090
windowWithHandle
IWindow 1051
windowWithOwner
IWindow 1070
windowWithParent
IWindow 1071
WM_COMMAND 210, 214
WM_CONTEXTMENU 576
WM_CONTROL 227
WM_INITMENU 572, 576
WM_MENUEND 572, 576
WM_MENUSELECT 572, 576
WM_SYSCOMMAND 210
wordBreak
IStaticText 913
wordWrap
IMultiLineEdit 666
wrapper
IMenuBar 559

X

x
ISWP 939
X/Motif messages
WM_CONTEXTMENU 576
XmCR_ACTIVATE 576
XmCR_MAP 576
XmCR_UNMA 576
XerrorCode
ICurrentThread 249
XmCR_ACTIVATE 576
XmCR_MAP 576
XmCR_UNMAP 576
xxxxxxx.inf xxv

Y

y
ISWP 939
yesNoButton
IMessageBox 611
yesNoCancelButton
IMessageBox 611

Communicating Your Comments to IBM

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume II

Version 3.5

Publication No. S33H-5040-00

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)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwt9)

Readers' Comments — We'd Like to Hear from You

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume II

Version 3.5

Publication No. S33H-5040-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

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-5040-00

IBM®

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

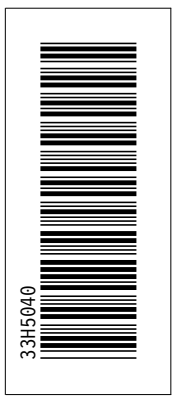
IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape | **Please do not staple** | Fold and Tape

S33H-5040-00

Cut or Fold
Along Line

IBM®

Part Number: 33H5040

Printed in U.S.A.