IBM C Set ++ for AIX: User Interface Class Library Reference Version 3.1

Document Number SC09-1971-00

IBM C Set ++ for AIX: User Interface Class Library

SC09-1971-00

Reference

Version 3.1

IBM C Set ++ for AIX: User Interface Class Library

SC09-1971-00

Reference

Version 3.1

Note

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

First Edition (October 1994)

This edition applies to Version 3.1 of IBM* C Set ++ for AIX (Program 5765-421) 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. Books are not stocked at the address given below.

A form for readers' comments is provided at the back of this book. 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 fax to (416) 448-6057 to the attention of the RCF Coordinator. Or, if you have access to Internet, 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 book.

Internet: **torrcf@vnet.ibm.com**
IBMLink*: **toribm(torrcf)**
IBM/PROFS*: **torolab4(torrcf)**
IBMMAIL: **ibmmail(caibmwt9)**

If you choose to respond through Internet, please include either your entire Internet network address or a postal address.

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 1994. 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

Notices xxvii
Trademarks xxvii

Part 1. Introduction 1

Chapter 1. About This Book 2
Who Should Read This Book 2
About the User Interface Class Library 2
How to Use This Book 3
How This Book is Organized 3
User Interface Class Library Conventions 4
Related Books 7

Part 2. Class Hierarchy by Category 11

Chapter 2. Class Hierarchy by Category 12
Application Classes 12
Data Types and Attributes Classes 13
Event Classes 14
Handler Classes 15
Support Classes 17
Window Classes 18

Part 3. Class Descriptions 21

Chapter 3. IAccelerator 22

Chapter 4. IAccelTblHandle 26

Chapter 5. IAnchorBlockHandle 27

Chapter 6. IApplication 28

Chapter 7. IBaseSpinButton 32

Chapter 8. IBaseSpinButton::Style 40

Chapter 9. IBitFlag 42

Chapter 10. IBitmapControl 45

Chapter 11. IBitmapControl::Style 51

© Copyright IBM Corp. 1994

Chapter 12. IBitmapHandle 53

Chapter 13. IButton 55

Chapter 14. IButton::Style 61

Chapter 15. ICanvas 63

Chapter 16. ICanvas::Style 70

Chapter 17. ICheckBox 71

Chapter 18. ICheckBox::Style 77

Chapter 19. ICnrControlList 79

Chapter 20. ICnrEmphasisEvent 80

Chapter 21. ICnrEnterEvent 83

Chapter 22. ICnrEvent 85

Chapter 23. ICnrHandler 87

Chapter 24. ICnrHelpEvent 91

Chapter 25. ICnrMenuHandler 93

Chapter 26. ICnrObjectSet 96

Chapter 27. ICnrQueryDeltaEvent 97

Chapter 28. ICnrScrollEvent 100

Chapter 29. IColor 103

Chapter 30. IComboBox 108

Chapter 31. IComboBox::Cursor 123

Chapter 32. IComboBox::Style 126

Chapter 33. ICommandEvent 128

Chapter 34. ICommandHandler 131

Chapter 35. IContainerColumn 135

Chapter 36. IContainerControl 149

Chapter 37. IContainerControl::Attribute 191

Chapter 38. IContainerControl::ColumnCursor 194

Chapter 39. IContainerControl::CompareFn 197

Chapter 40. IContainerControl::FilterFn 199

Chapter 41. IContainerControl::Iterator 200

Chapter 42. IContainerControl::ObjectCursor 201

Chapter 43. IContainerControl::Style 204

Chapter 44. IContainerControl::TextCursor 206

Chapter 45. IContainerObject 209

Chapter 46. IContextHandle 219

Chapter 47. IControl 220

Chapter 48. IControl::Style 225

Chapter 49. IControlEvent 227

Chapter 50. ICritSec 229

Chapter 51. ICurrentApplication 231

Chapter 52. ICurrentThread 234

Chapter 53. IDeviceColor 241

Chapter 54. IDisplayHandle 243

Chapter 55. IDynamicLinkLibrary 244

Chapter 56. IEditHandler 247

Chapter 57. IEditVerifyEvent 249

Chapter 58. IEditVerifyHandler 253

Chapter 59. IEntryField 256

Chapter 60. IEntryField::Style 271

Chapter 61. IEnumHandle 274

Chapter 62. IEvent 275

Chapter 63. IEventData 282

Chapter 64. IFileDialog 286

Chapter 65. IFileDialog::Settings 294

Chapter 66. IFileDialog::Style 299

Chapter 67. IFileDialogEvent 301

Chapter 68. IFileDialogHandler 303

Chapter 69. IFocusHandler 307

Chapter 70. IFont 310

Chapter 71. IFontDialog 322

Chapter 72. IFontDialog::Settings 329

Chapter 73. IFontDialog::Style 332

Chapter 74. IFontDialogHandler 334

Chapter 75. IFrameEvent 336

Chapter 76. IFrameExtension 338

Chapter 77. IFrameExtensions 344

Chapter 78. IFrameFormatEvent 345

Chapter 79. IFrameHandler 347

Chapter 80. IFrameWindow 352

Chapter 81. IFrameWindow::Style 376

Chapter 82. IGraphicPushButton 380

Chapter 83. IGraphicPushButton::Style 388

Chapter 84. IGroupBox 390

Chapter 85. IGroupBox::Style 396

Chapter 86. IGUIColor 397

Chapter 87. IHandle 400

Chapter 88. IHandler 402

Chapter 89. IHelpErrorEvent 405

Chapter 90. IHelpHandler 408

Chapter 91. IHelpHypertextEvent 413

Chapter 92. IHelpMenuBarEvent 415

Chapter 93. IHelpNotifyEvent 416

Chapter 94. IHelpSubitemNotFoundEvent 418

Chapter 95. IHelpTutorialEvent 420

Chapter 96. IHelpWindow 422

Chapter 97. IHelpWindow::Settings 431

Chapter 98. IIconControl 433

Chapter 99. IIconControl::Style 439

Chapter 100. IInfoArea 441

Chapter 101. IKeyboardEvent 449

Chapter 102. IKeyboardHandler 455

Chapter 103. IListBox 458

Chapter 104. IListBox::Cursor 474

Chapter 105. IListBox::Style 477

Chapter 106. IMenu 479

Chapter 107. IMenu::Cursor 495

Chapter 108. IMenu::Style 497

Chapter 109. IMenuBar 499

Chapter 110. IMenuBar::Style 504

Chapter 111. IMenuEvent 506

Chapter 112. IMenuHandler 508

Chapter 113. IMenuItem 512

Chapter 114. IMenuItem::Attribute 521

Chapter 115. IMenuItem::Style 523

Chapter 116. IMessageBox 525

Chapter 117. IMessageBox::Style 529

Chapter 118. IMessageQueueHandle 532

Chapter 119. IModuleHandle 534

Chapter 120. IMouseClickEvent 535

Chapter 121. IMouseClickHandler 539

Chapter 122. IMultiCellCanvas 541

Chapter 123. IMultiCellCanvas::Style 555

Chapter 124. IMultiLineEdit 557

Chapter 125. IMultiLineEdit::Style 577

Chapter 126. INotebook 579

Chapter 127. INotebook::Cursor 597

Chapter 128. INotebook::PageSettings 600

Chapter 129. INotebook::PageSettings::Attribute 605

Chapter 130. INotebook::Style 607

Chapter 131. INotifier 610

Chapter 132. INumericSpinButton 612

Chapter 133. INumericSpinButton::Style 619

Chapter 134. IObjectWindow 621

Chapter 135. IOutlineBox 624

Chapter 136. IOutlineBox::Style 630

Chapter 137. IPageEvent 632

Chapter 138. IPageHandle 634

Chapter 139. IPageHandler 635

Chapter 140. IPageHelpEvent 638

Chapter 141. IPageRemoveEvent 640

Chapter 142. IPageSelectEvent 642

Chapter 143. IPaintEvent 644

Chapter 144. IPaintHandler 647

Chapter 145. IPair 649

Chapter 146. IPoint 655

Chapter 147. IPointerHandle 657

Chapter 148. IPopUpMenu 659

Chapter 149. IPresSpaceHandle 664

Chapter 150. IPrivateResource 666

Chapter 151. IProcedureAddress 668

Chapter 152. IProcessId 670

Chapter 153. IProfile 671

Chapter 154. IProfile::Cursor 675

Chapter 155. IProfileHandle 677

Chapter 156. IProgressIndicator 679

Chapter 157. IProgressIndicator::Style 695

Chapter 158. IPushButton 697

Chapter 159. IPushButton::Style 705

Chapter 160. IRadioButton 707

Chapter 161. IRadioButton::Style 713

Chapter 162. IRange 715

Chapter 163. IRectangle 717

Chapter 164. IRefCounted 725

Chapter 165. IReference 727

Chapter 166. IRegionHandle 729

Chapter 167. IResizeEvent 730

Chapter 168. IResizeHandler 732

Chapter 169. IResource 734

Chapter 170. IResourceId 736

Chapter 171. IResourceLibrary 738

Chapter 172. IResourceLock 744

Chapter 173. IScrollBar 746

Chapter 174. IScrollBar::Style 756

Chapter 175. IScrollEvent 758

Chapter 176. IScrollHandler 760

Chapter 177. ISelectHandler 764

Chapter 178. ISemaphoreHandle 767

Chapter 179. ISetCanvas 768

Chapter 180. ISetCanvas::Style 778

Chapter 181. ISettingButton 781

Chapter 182. ISharedResource 786

Chapter 183. IShowListHandler 788

Chapter 184. ISize 790

Chapter 185. ISlider 792

Chapter 186. ISlider::Style 801

Chapter 187. ISpinButton 803

Chapter 188. ISpinButton::Cursor 817

Chapter 189. ISpinButton::Style 819

Chapter 190. ISpinHandler 821

Chapter 191. ISplitCanvas 824

Chapter 192. ISplitCanvas::Style 834

Chapter 193. IStaticText 836

Chapter 194. IStaticText::Style 846

Chapter 195. IStringHandle 849

Chapter 196. ISubmenu 850

Chapter 197. ISubmenu::Cursor 859

Chapter 198. ISWP 861

Chapter 199. ISWPArray 863

Chapter 200. ISystemBitmapHandle 865

Chapter 201. ISystemMenu 867

Chapter 202. ISystemPointerHandle 872

Chapter 203. ITextControl 874

Chapter 204. ITextSpinButton 878

Chapter 205. ITextSpinButton::Cursor 886

Chapter 206. ITextSpinButton::Style 888

Chapter 207. IThread 890

Chapter 208. IThread::Cursor 900

Chapter 209. IThreadFn 902

Chapter 210. IThreadId 903

Chapter 211. IThreadMemberFn 904

Chapter 212. ITitle 906

Chapter 213. IViewPort 914

Chapter 214. IViewPort::Style 922

Chapter 215. IWindow 924

Chapter 216. IWindow::ChildCursor 954

Chapter 217. IWindow::ExceptionFn 956

Chapter 218. IWindow::Style 957

Chapter 219. IWindowHandle 960

Appendix A. New Color Support 963

Appendix B. Task and Samples Cross-Reference Table 973

Appendix C. Handlers for Controls Cross-Reference Table 975

Appendix D. Classes and Their Styles Cross-Reference Table 977

Appendix E. Classes and Header Files Cross-Reference Tables 984

Appendix F. Obsolete and Ignored Members Cross-Reference Tables . . . 997

Glossary 1012

Index 1018

Notices

Any reference to an IBM product, program, or service is not intended to state or imply that only an IBM product, program, or service can be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights can 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 book. The furnishing of this book does not give you any license to these patents. You can send license inquiries, in writing, to:

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

IBM can change this book, the product described herein, or both. These changes will be incorporated in new editions of the book.

Trademarks

The following terms, denoted by an asterisk (*) on their first occurrence in this publication, are trademarks of IBM Corporation in the United States or other countries:

AIX
AIX/6000
BookManager
C Set ++
Common User Access
CUA
IBM
IBMLink
Operating System/2
OS/2
Presentation Manager
PROFS
WorkFrame/2

The following terms, denoted by a double asterisk (**) on their first occurrence in this publication, are trademarks of other companies:

ANSI	Trademark of the American National Standards Institute
AT&T	Trademark of the American Telephone and Telegraph Company
Motif	Trademark of the Open Software Foundation, Inc.

© Copyright IBM Corp. 1994

Part 1. Introduction

© Copyright IBM Corp. 1994

Chapter 1. About This Book

The *IBM C Set ++ for AIX: User Interface Class Library Reference* (hereafter called *User Interface Class Library Reference*) provides an alphabetical reference of classes for the User Interface Class Library, which is part of the C Set ++ for AIX product.

Who Should Read This Book

This book is intended for application programmers who want to develop portable C++ applications using the User Interface Class Library product. You should be familiar with the IBM AIX* Version 4.1 Operating System and understand the C++ programming language.

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

About the User Interface Class Library

The User Interface Class Library is one of the class libraries included in the IBM C Set ++ for AIX product. The User Interface Class Library is a C++ class library that simplifies how you develop Motif** applications so that you can take advantage of Motif 1.2 features. The User Interface Class Library provides classes that you can use in your applications and reuse to extend your applications.

The primary purpose of the User Interface Class Library is to help you develop applications that are portable between the AIX operating system and Operating System/2* (OS/2*).

The User Interface Class Library contains over 260 classes and over 2600 member functions. To assist you in learning about the classes and to guide you as you start developing applications, we put the classes into the following basic categories:

- Application
- Data Types
- Event
- Handler
- Settings and Styles
- Support
- Window

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

Data Type Classes: Model basic data types, such as 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.

© Copyright IBM Corp. 1994

Event and Handler 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 control has default processing that, in most cases, you can override using the handler classes.

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.

Settings and Style Classes: Change the appearance or behavior of window classes. For example, the IFileDialog class lets you specify the default window title to be displayed through its settings class. Setting the text alignment, specifying a minimize button on an application main window, or whether a window is initially visible are three examples of styles that you can set.

Support Classes: Work with other classes. An example of these classes is cursors.

Window Classes: Encapsulate the basic graphical building blocks that are used to construct application windows. These range from simple graphical objects like title bars, which display the title of the window, to complex objects like containers, which can contain other objects and provide different views of those objects. Window classes support both parent and owner windows. This allows window position and appearance (parent windows) to be separated from event handling (owner windows).

See "How to Use This Book" for tips on using this document.

How to Use This Book

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

How This Book is Organized

In "Related Books" on page 7 you can find books related to the C Set ++ for AIX User Interface Class Library and the C/C++ Tools User Interface Class Library, C and C++ books, as well as other useful publications.

Chapter, "Class Hierarchy by Category" on page 12, groups User Interface Class Library classes into major categories. You can use this information as a guide when choosing classes.

You can quickly locate and learn more about any User Interface Class Library class using the alphabetical reference in Part 3, "Class Descriptions." This section comprises the majority of this document.

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

Portability Considerations
Motif Information
Presentation Manager Information

These sections contain information that is specific to a particular platform or suggestions for developing portable applications.

Note that some members have conditions. 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 both AIX and OS/2, while another implementation is for OS/2 only. For more information, see Part 3, "Class Descriptions."

Appendix A, "New Color Support" lists all the User Interface Class Library classes with new color member functions.

Appendix B, "Task and Samples Cross-Reference Table" contains the table that cross-references tasks to the samples and examples in the *User Interface Class Library User's Guide*.

Appendix C, "Handlers for Controls Cross-Reference Table" contains a list of handlers and the controls to which you can attach them.

Appendix D, "Classes and Their Styles Cross-Reference Table" contains a list of styles, showing the styles each class has, and whether the User Interface Class Library for AIX supports the style.

Appendix E, "Classes and Header Files Cross-Reference Tables" 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.

Appendix F, "Obsolete and Ignored Members Cross-Reference Tables" contains two cross-reference tables. The first table lists obsolete classes and members in the User Interface Class Library for AIX. The second table lists classes and members that the User Interface Class Library for AIX does not support and subsequently ignores.

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

File Names

All files provided by the User Interface Class Library begin with the letter "I" for IBM, for example, iapp.hpp. The following table lists 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.

Refer to the *User Interface Class Library Reference* for cross-reference tables for the header files and the classes they contain.

The following table lists some other file names, files 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 as `const` references or 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 OS/2 and AIX. 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 `== true`.

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

  ```
  ISize size() const;                //Returns an 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:

  ```
  aWindow.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.

A Note about Samples and Examples

This *User Interface 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 `/usr/lpp/xlC/iclui/samples` directory. Examples, on the other hand, exist only in the *User Interface 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.

Chapter 2. Related Books

This section lists the related books comprising the C Set ++ library referenced in this book. The list of related IBM product books and other related books is not exhaustive, but it should be adequate for most C Set ++ users.

C Set ++ for AIX Library

Title	Order Number
C Set ++ for AIX: Application Support Class Library Reference	SC09-1729
C Set ++ for AIX: Collection Class Library Reference	SC09-1602
C Set ++ for AIX: C++ Language Reference	SC09-1606
C Set ++ for AIX: HeapView Debugger User's Guide	SC09-1728
C Set ++ for AIX: Installation Instructions	GC09-1608
C Set ++ for AIX: Source Code Browser User's Guide	SC09-1603
C Set ++ for AIX: Standard Class Library Reference	SC09-1604
C Set ++ for AIX: User's Guide	SC09-1605
C Set ++ for AIX: User Interface Class Library Reference Volume 1	SC09-1978
C Set ++ for AIX: User Interface Class Library Reference Volume 2	SC09-2085
C Set ++ for AIX: User Interface Class Library User's Guide	SC09-1977

C/C++ Tools for OS/2 Library

Title	Order Number
C Set ++ Library — Group 1	61G1439
C/C++ Tools: Browser Introduction	61G1397
C/C++ Tools: Debugger Introduction	61G1184
C/C++ Tools: Developer's Toolkit for OS/2: Getting Started	10G6199
C/C++ Tools: Execution Trace Analyzer Introduction	61G1398
C/C++ Tools: Programming Guide	61G1181
C/C++ Tools: Read Me First	82G2871
C/C++ Tools: Reference Summary	61G1441
C/C++ Tools: User Interface Class Library User's Guide	82G3743
C/C++ Tools: WorkFrame/2 Introduction V1.1*	61G1428
C/C++ Tools: WorkFrame/2 Introduction V2.1*	82G3740
Developer's Toolkit for OS/2 2.1 Getting Started	60G9284
MMPM/2 Getting Started	61G1634

Title	Order Number
C Set ++ Library — Group 2	61G1440
C/C++ Tools: C Language Reference	61G1399

© Copyright IBM Corp. 1994

Title	Order Number
C/C++ Tools: C Library Reference	61G1183
C/C++ Tools: Collection Class Library Reference	61G1178
C/C++ Tools: C++ Language Reference	61G1185
C/C++ Tools: Standard Class Library Reference	61G1180
C/C++ Tools: User Interface Class Library Reference	82G3738

C and C++ Books and Standards

The C++ language was developed by Bjarne Stroustrup of AT&T** Bell Laboratories. For more information about the history of the language, refer to books and papers written by him.

The following are some non-IBM publications that are generally available. This is not an exhaustive list. IBM does not specifically recommend these books, and other publications may be available in your local library or bookstore.

- Ellis, Margaret A. and Bjarne Stroustrup, *The Annotated C++ Reference Manual*. Reading, Mass.: Addison-Wesley Publishing Company, 1990.
- Lippman, Stanley B., *C++ Primer*. 2d ed. Reading, Mass.: Addison-Wesley Publishing Company, 1991.
- Stroustrup, Bjarne, *The C++ Programming Language*. 2d ed. Reading, Mass.: Addison-Wesley Publishing Company, 1991.

You might also want to refer to the following standards:

- *Working Paper for Draft Proposed American National Standard for Information Systems — Programming Language C++, (X3J16/92-0060)* presents an American National Standards Institute (ANSI**) committee's draft of a C++ language standard.
- *ANSI/ISO-IEC 9899-1990[1992]* presents the ANSI and the International Standards Organization (ISO) standard for the C language.

  Note: This document officially replaces *American National Standard for Information Systems—Programming Language C (X3.159-1989)* as the ANSI C Standard.
- *ISO/EEC 9899:1990(E)* presents the ISO C language standard.
- *Federal Information Processing Standards Publication C, FIPS PUB 160* presents the Federal Information Processing Standards (FIPS) for the C language.

IBM OS/2 2.1 Books

The following books are available only as part of OS/2 2.1 or the Developer's Toolkit products.

Title	Order Number
OS/2 2.1 Documentation Only	S61G-0905
OS/2 2.1 Bidirectional Support	S71G-4114
OS/2 2.1 Book Catalog	S61G-0706
OS/2 2.1 Command Reference	S71G-4112
OS/2 2.1 Information and Planning Guide	S61G-0913
OS/2 2.1 Installation Guide	S61G-0704
OS/2 2.1 Keyboard and Codepages	S71G-4113
OS/2 2.1 Quick Reference	S61G-0713
OS/2 2.1 Using the Operating System	S61G-0703

IBM OS/2 2.1 Technical Library

The following books comprise the OS/2 2.1 Technical Library.

Title	Order Number
Control Program Programming Reference	S10G-6263
Information Presentation Facility Guide and Reference	S10G-6262
OS/2 2.1 PM Programming Reference Volume 1	S10G-6264
OS/2 2.1 PM Programming Reference Volume 2	S10G-6265
OS/2 2.1 PM Programming Reference Volume 3	S10G-6272
Physical Device Driver Reference for OS/2	S10G-6266
Presentation Device Driver Reference for OS/2	S10G-6267
Virtual Device Driver Reference	S10G-6310

IBM BookManager* READ/2 Books

Title	Order Number
IBM BookManager READ/2: Displaying Online Books	SB35-0801
IBM BookManager READ/2: General Information	GB35-0800
IBM BookManager READ/2: Getting Started and Quick Reference	SX76-0146
IBM BookManager READ/2: Installation	GX76-0147

Style Guides

International Business Machines. *Object-Oriented Interface Design: IBM Common User Access* Guidelines*. Carmel: Que Corporation, 1992.

OSF/Motif Style Guide, Release 1.2. Englewood Cliffs, New Jersey: PTR Prentice-Hall, Inc., 1989.

X Window System Series Books

The following are O'Reilly and Associates, Inc. publications. IBM does not specifically recommend these books.

Volume 0: X Protocol Reference Manual. 3d ed. 1992.

Volume 1: Xlib Programming Manual. 3d ed. 1992.

Volume 2: Xlib Reference Manual. 3d ed. 1992.

Volume 3M: X Window System User's Guide. 2d ed. 1993.

Volume 4M: X Toolkit Intrinsics Programming Manual. 2d (OSF/Motif) ed. 1992.

Volume 5: X Toolkit Intrinsics Reference Manual. 3d ed. 1992.

Volume 6A: Motif Programming Manual. (OSF/Motif edition.) 1993.

Volume 6B: Motif Reference Manual. (OSF/Motif edition.) 1993.

Part 2. Class Hierarchy by Category

© Copyright IBM Corp. 1994

Chapter. Class Hierarchy by Category

The User Interface Class Library contains over 260 classes and over 2600 member functions. To assist you in learning about the classes and to guide you as you develop applications, the classes are divided into categories.

Note: <CCL> refers to the Collection Class Library; <ASCL> refers to the Application Support Class Library.

Application Classes

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

```
IBase <ASCL>
   ICritSec
   IProcedureAddress
   IReference
   IResourceId
   IVBase <ASCL>
      IApplication
         ICurrentApplication
      IProfile
      IRefCounted
         IThreadFn
            IThreadMemberFn
      IResource
         IPrivateResource
         ISharedResource
      IResourceLibrary
         IDynamicLinkLibrary
      IResourceLock
      IThread
         ICurrentThread
```


© Copyright IBM Corp. 1994

Data Types and Attributes Classes

Model basic data types, such as 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.

```
IBase <ASCL>
    IVBase <ASCL>
        IColor
            IDeviceColor
            IGUIColor
    IHandle
        IAccelTblHandle
        IAnchorBlockHandle
        IBitmapHandle
            ISystemBitmapHandle
        IContextHandle
        IDisplayHandle
        IEnumHandle
        IMessageQueueHandle
        IModuleHandle
        IPageHandle
        IPointerHandle
            ISystemPointerHandle
        IPresSpaceHandle
        IProcessId
        IProfileHandle
        IRegionHandle
        ISemaphoreHandle
        IStringHandle
        IThreadId
        IWindowHandle
    IPair
        IPoint
        IRange
        ISize
    IPictureData
    IPictureElement
    IRectangle
    IVBase <ASCL>
        IFont
```

Event 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.

```
IBase <ASCL>
   IEventData
   IEventParameter1
   IEventParameter2
   IEventResult
   IHighEventParameter
   ILowEventParameter
   IVBase <ASCL>
      IEvent
         ICommandEvent
         IControlEvent
            ICnrEvent
               ICnrEmphasisEvent
               ICnrEnterEvent
               ICnrHelpEvent
               ICnrQueryDeltaEvent
               ICnrScrollEvent
            IEditVerifyEvent
            IPageEvent
               IPageHelpEvent
               IPageRemoveEvent
               IPageSelectEvent
         IFileDialogEvent
         IFrameEvent
            IFrameFormatEvent
         IHelpErrorEvent
         IHelpHypertextEvent
         IHelpMenuBarEvent
         IHelpNotifyEvent
         IHelpSubitemNotFoundEvent
         IHelpTutorialEvent
         IKeyboardEvent
         IMenuEvent
         IMouseClickEvent
         IPaintEvent
         IResizeEvent
         IScrollEvent
```

Handler Classes

Each control has default processing that, in most cases, you can override using the handler classes.

Use handlers to override or augment a control's default behavior.

IBase <ASCL>
- IVBase <ASCL>
  - IHandler
    - ICnrHandler
    - ICommandHandler
    - IEditHandler
    - IEditVerifyHandler
      - IPictureVerifyHandler
    - IFileDialogHandler
    - IFocusHandler
    - IFontDialogHandler
    - IFrameHandler
    - IHelpHandler
    - IKeyboardHandler
    - IMenuHandler
      - ICnrMenuHandler
      - IInfoArea
    - IMouseClickHandler
    - IPageHandler
    - IPaintHandler
    - IResizeHandler
    - IScrollHandler
    - ISelectHandler
    - IShowListHandler
    - ISpinHandler

Settings and Styles Classes

The settings and style classes change the appearance or behavior of window classes.

IBase <ASCL>
- IBitFlag
  - IBaseSpinButton::Style
  - IBitmapControl::Style
  - IButton::Style
  - ICanvas::Style
  - ICheckBox::Style
  - IComboBox::Style
  - IContainerControl::Attribute
  - IContainerControl::Style
  - IControl::Style
  - IEntryField::Style
  - IFileDialog::Style
  - IFontDialog::Style
  - IFrameWindow::Style
  - IGraphicPushButton::Style
  - IGroupBox::Style
  - IIconControl::Style
  - IListBox::Style
  - IMenu::Style
  - IMenuBar::Style
  - IMenuItem::Attribute
  - IMenuItem::Style
  - IMessageBox::Style
  - IMultiCellCanvas::Style
  - IMultiLineEdit::Style
  - INotebook::PageSettings::Attribute
  - INotebook::Style
  - INumericSpinButton::Style
  - IOutlineBox::Style
  - IProgressIndicator::Style
  - IPushButton::Style
  - IRadioButton::Style
  - IScrollBar::Style
  - ISetCanvas::Style
  - ISlider::Style
  - ISpinButton::Style
  - ISplitCanvas::Style
  - IStaticText::Style
  - ITextSpinButton::Style
  - IViewPort::Style
  - IWindow::Style
- IFileDialog::Settings
- IHelpWindow::Settings
- IFontDialog::Settings
- IVBase <ASCL>
  - INotebook::PageSettings

Support Classes

The support classes work with other classes. This category includes nested classes.

IBase <ASCL>
- IAccelerator
- IFrameExtension
- IMenuItem
- ISWP
- ISWPArray
- IVBase <ASCL>
  - IComboBox::Cursor
  - IContainerColumn
  - IContainerControl::ColumnCursor
  - IContainerControl::CompareFn
  - IContainerControl::FilterFn
  - IContainerControl::Iterator
  - IContainerControl::ObjectCursor
  - IContainerControl::TextCursor
  - IContainerObject
  - IListBox::Cursor
  - IMenu::Cursor
  - INotebook::Cursor
  - IProfile::Cursor
  - ISpinButton::Cursor
  - ISubmenu::Cursor
  - ITextSpinButton::Cursor
  - IThread::Cursor
  - IWindow::ChildCursor

ISet <CCL>
- ICnrControlList

ISequence <CCL>
- ICnrObjectSet
- IFrameExtensions
- IPictureElementList

Window Classes

Encapsulate the basic graphical building blocks that are used to construct application windows. These range from simple graphical objects like title bars, which display the title of the window, to complex objects like containers, which can contain other objects and provide different views of those objects. Window classes support both parent and owner windows. This allows window position and appearance (parent windows) to be separated from event handling (owner windows).

```
IBase <ASCL>
   IVBase <ASCL>
      IMessageBox
      IWindow::ExceptionFn
      INotifier
         IWindow
            IControl
               IBaseSpinButton
                  INumericSpinButton
                  ITextSpinButton
               ICanvas
                  IMultiCellCanvas
                  ISetCanvas
                  ISplitCanvas
                  IViewPort
               IContainerControl
               IListBox
               INotebook
               IOutlineBox
               IProgressIndicator
                  ISlider
               IScrollBar
               ISpinButton
               ITextControl
                  IButton
                     IPushButton
                        IGraphicPushButton
                     ISettingButton
                        ICheckBox
                        IRadioButton
                  IEntryField
                     IComboBox
                  IGroupBox
                  IMultiLineEdit
                  IStaticText
                     IBitmapControl
                        IIconControl
                     IInfoArea
                  ITitle
```

IFrameWindow
    IFileDialog
    IFontDialog
IHelpWindow
IMenu
    IMenuBar
    IPopUpMenu
    ISubmenu
    ISystemMenu
IObjectWindow

Part 3. Class Descriptions

The User Interface Class Library contains members that are labeled with a condition in an effort to clarify when you can use them. The condition immediately precedes the member it labels and the condition *only* labels the member function immediately following it. It is possible for an overloaded member function to have a different condition for each implementation. It is also possible for a member function to have multiple conditions.

Conditions

1. (Ignored on AIX) The following member performs no operation on the AIX platform.
2. (OS/2 only) The following implementation of this member in this class is only valid on the OS/2 platform. In AIX, there might be an inherited member that you can use. See the inherited members table at the end of the class' member section.
3. (AIX only) The following implementation of this member in this class is only valid on the AIX platform. In OS/2, there might be an inherited member that you can use. See the inherited members table at the end of the class' member section.
4. (Typically for internal use) Although the following member is public or protected, treat it as private and do not use it. The User Interface Class Library provides this member for its internal implementation only.

© Copyright IBM Corp. 1994

Chapter 4. IAccelerator

Derivation

IBase
  IAccelerator

Header File: iaccel.hpp

Objects of the IAccelerator class add 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.

Portability Considerations: In Presentation Manager, the User Interface Class Library constructs accelerator table objects from resources and installs the objects on entire windows. Motif has no equivalent concept. See AIX-Specific Information for further details.

Motif Information: In AIX, the User Interface Class Library does not create or maintain IAccelTblHandle objects. Motif has no equivilant concept. In Motif shortcut keys must be installed on individual widgets. You cannot activate shortcut keys for a window or an entire application, but only for menu items within a window's menu system.

The User Interface Class Library activates accelerators by installing each shortcut key listed in the accelerator resource on a menu item whose IDmatches the accelerator command value. If no match is found, the shortcut key is not installed and thus will not activate any command actions when pressed. This implies 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 shortcut keys have been installed, they cannot be removed.

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

In AIX, you cannot do the following:

- Attach accelerators to push buttons, windows, or applications.
- Create accelerator table objects.
- Use these member functions:
  - IAccelerator::handle, which always returns 0
  - IAccelerator::isSet
  - IAccelerator::remove
  - IAccelerator::reset

© Copyright IBM Corp. 1994

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

Presentation Manager Information: Use this class to associate shortcut keys for a single window or an entire application. When you do, ensure that the frame window associated with the table has the keyboard focus.

Constructors

Condition: (2) OS/2 Only

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

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

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

You can construct objects of this class from the following:

- An IAccelTblHandle (see page 26)
- An IResourceId (see page 736)

The constructors accept an optional parameter, *owner*, that is an IWindow (see page 924). If you specify an IWindow, the accelerator is only in effect when the specified IWindow object has the focus. If you do not specify *owner* on the constructor, the IAccelerator applies to all windows in the application but has no effect on other applications that are currently running.

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

```
IAccelerator* pac1 = new IAccelerator(myAccelRes);
IAccelTblHandle haccel1 = pac1->accel();
delete pac1;
pac1 = New IAccelerator(haccel1); //haccel1 no longer valid
```

Public Members

handle

Condition: (1) Ignored on AIX

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

Returns the IAccelTblHandle (see page 26) of the accelerator in effect.

isSet

Condition: (1) Ignored on AIX

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

If the accelerator is in effect, true is returned.

owner

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

Returns the IWindow (see page 924) to which the accelerator applies.

remove

Condition: (1) Ignored on AIX

```
IAccelerator &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 (see page 26) after removing it, an exception is thrown.

reset

Condition: (1) Ignored on AIX

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

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

set

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

Condition: (1) Ignored on AIX

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

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

Changes to the new accelerator.

haccel Accelerator handle.

accelResId Accelerator resource ID.

Presentation Manager Information: Saves the current accelerator and changes to the new accelerator.

Figure 1. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 5. IAccelTblHandle

Derivation

IBase
  IHandle
    IAccelTblHandle

Header File: ihandle.hpp

Objects of the IAccelTblHandle class access accelerator tables. Accelerator tables are also referred to as shortcut tables. The class IAccelerator (see page 22) describes accelerator tables.

Motif Information: AIX does not support this class.

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

Constructors

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

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

Public Members

operator Value

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

Returns the IAccelTblHandle value.

Figure 2. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 6. IAnchorBlockHandle

Derivation

IBase
  IHandle
    IAnchorBlockHandle

Header File: ihandle.hpp

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

Motif Information: AIX does not support this class.

Presentation Manager Information: IAnchorBlockHandle is an alias for the OS/2 Programmer's Toolkit type HAB.

Constructors

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

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

Public Members

operator Value

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

Returns the IAnchorBlockHandle value.

Figure 3. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 7. IApplication

Derivation

IBase
  IVBase
    IApplication

Header File: iapp.hpp

Objects of the IApplication class represent processes. The User Interface Class Library supports only the currently executing application, the single object of the derived class ICurrentApplication (see page 231). IApplication::current provides access to that object.

The IApplication class maintains a static pointer to the C++ object representing the currently executing application. IApplication does so to implement the static member function current (see page 29), 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.

Constructors

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

You can only construct objects of this class with the process identifier for the process the object will represent. The constructor is protected; only the derived class ICurrentApplication (see page 231) utilizes it.

Public Members

adjustPriority

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

Adjusts the priority level of all the application's threads by some amount. An optional flag indicates whether or not to also adjust the priority of threads of descendent processes.

Motif Information: 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.

© Copyright IBM Corp. 1994

asDebugInfo

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

Returns a representation of the application as debug information. The representation is returned as an IString with the contents `IApplication(IVBase(@addr),id=pid)` which represent the following:

addr The address of the object (in hexadecimal)

pid The process ID (in hexadecimal)

asString

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

Returns the string "IApplication(process identifier value)."

current

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

Returns a reference to the current application, which is an object of the class ICurrentApplication (see page 231).

currentPID

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

Returns the current process identifier.

id

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

Returns the object's process identifier value.

setPriority

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

Sets the priority (see page 30) class and level of all the application's threads to the specified value. An optional *Boolean* parameter specifies whether the library should also modify the priority of descendent processes.

Motif Information: 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.

Figure 4. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

setId

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

Sets the process identifier. Call this function in the implementation of a derived class when you start a process.

Enumerations

PriorityClass

```
public:
enum PriorityClass
  {
  noChange,
  idleTime,
  regular,
  timeCritical,
  foregroundServer,
  };
```

These enumerators specify one of the priority classes, such as when using setPriority: (see page 29)

noChange
Specifies no change to the thread priority.

idleTime
Specifies the lowest priority. This priority only gets CPU 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 should take precedence other processes on the server.

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

Presentation Manager Information: For additional information, see the *OS/2 2.1 Technical Library Programming Guide Volume I*.

Chapter 8. IBaseSpinButton

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          IBaseSpinButton
```

Header File: ispinbas.hpp

IBaseSpinButton is an abstract base class for spin button controls. Use the IBaseSpinButton derived classes, INumericSpinButton (see page 612) and ITextSpinButton (see page 878), to create and manage numeric and text spin button controls.

Motif Information: Spin buttons created during object construction of IBaseSpinButton derived classes use several 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, you should 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 prior to the default handler.

Nested Classes

IBaseSpinButton::Style

Constructors

```
protected:
IBaseSpinButton();
```

You cannot construct objects of this class because it is an abstract base class.

© Copyright IBM Corp. 1994

Public Members

addBorder

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

Adds a border to the spin button.

alignment

Condition: (1) Ignored on AIX

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

Returns the current alignment in the spin field of this spin button object. The returned value is an enumerator provided by Alignment (see page 39).

Motif Information: This function always returns left.

disableDataUpdate

```
virtual IBaseSpinButton &disableDataUpdate();
```

Disables direct editing of the spin field data by the end user.

disableFastSpin

```
virtual IBaseSpinButton &disableFastSpin();
```

Disables fast spinning of the spin button.

enable

Condition: (2) OS/2 Only

```
virtual IBaseSpinButton &enable(Boolean enable = true);
```

Enables or disables the window from accepting keyboard and mouse input.

enableDataUpdate

```
virtual IBaseSpinButton &enableDataUpdate(
                            Boolean writeable = true);
```

Enables or disables direct editing of spin field data by the end user.

enableFastSpin

```
virtual IBaseSpinButton &enableFastSpin(
                            Boolean fast = true);
```

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.

foregroundColor

Condition: (2) OS/2 Only

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

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

hasBorder

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

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

isFastSpinEnabled

```
Boolean isFastSpinEnabled() const;
```

If the spin speed is doubled every two seconds, true is returned.

isMaster

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

If the spin button is a master, true is returned.

isServant

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

If the spin button is a servant, true is returned.

isSpinFieldValid

```
virtual Boolean isSpinFieldValid(
                  Boolean caseSensitive) const = 0;
```

This is a pure virtual function that derived classes use to determine if the contents of the spin field are valid.

isWriteable

```
Boolean isWriteable() const;
```

If the user cannot type in the spin field, true is returned.

limit

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

Returns the number of characters permitted in the spin field.

moveSizeTo

Condition: (3) AIX only

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

Changes the position and size of the spin button window using a coordinate system that is lower-left origin based.

aRectangle
A rectangle for the spin button window.

removeBorder

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

Removes the border from the spin button.

setAlignment

Condition: (1) Ignored on AIX

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

Changes the text or number alignment in the spin field.

alignment The default is left. Use the enumeration Alignment (see page 39) to specify the alignment of the text or number.

setLimit

```
virtual IBaseSpinButton &setLimit(
                            unsigned long aNumber = 255);
```

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.

setMaster

```
virtual IBaseSpinButton &setMaster(IBaseSpinButton &master);
```

Defines a servant spin button's master. The spin button for which you call this function *must* be a servant. If it is not, this function throws an IInvalidParameter exception.

spinDown

```
virtual IBaseSpinButton &spinDown(unsigned long spinBy = 1);
```

Spins the button down the specified number of times.

spinUp

```
virtual IBaseSpinButton &spinUp(unsigned long spinBy = 1);
```

Spins the button up the specified number of times.

topHandle

Condition: (3) AIX only

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

Returns the top-most manager widget of the spin button.

Figure 5. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow

Figure 5. Inherited Public Members

Member function	Class
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow

Figure 5. Inherited Public Members

Member function	Class
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow

Figure 5. Inherited Public Members

Member function	Class
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the recommended minimum size of this spin button control. The size is based on the font and the text limit.

initialize

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

Condition: (2) OS/2 Only

```
virtual void initialize();
```

Used by several of the constructors to create a spin button control.

registerCallbacks

Condition: (3) AIX only

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

Registers Motif callbacks for the widgets created during construction of objects of derived classes.

unregisterCallbacks

Condition: (3) AIX only

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

Unregisters Motif callbacks for the widgets created during construction of objects of derived classes.

Figure 6. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow

Figure 6. Inherited Protected Members

Member function	Class
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow

Figure 6. Inherited Protected Members

Member function	Class
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

Condition: (1) Ignored on AIX

```
public:
enum Alignment
  {
  left,
  right,
  center,
  };
```

Use these enumerators to specify the alignment of text in the spin field:

left
Left-justifies the text or number.

center
Centers the text or number.

right
Right-justifies the text or number.

Motif Information: The AIX release ignores the Alignment enumeration. All spin buttons have left alignment.

Chapter 9. IBaseSpinButton::Style

Derivation

IBase
  IBitFlag
    IBaseSpinButton::Style

Header File: ispinbas.hpp

The nested class IBaseSpinButton::Style provides a set of valid spin button styles. You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

centerAlign
Centers the text or number.

classDefaultStyle
Specifies the original default style for this class, which is master | leftAlign | IWindow::visible.

fastSpin
Enables the spin button to increase its spin speed with time. The speed doubles every two seconds.

leftAlign
Left-justifies the text or number.

master
Specifies 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

When a 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.

noBorder
Suppresses the drawing of a border.

readOnly
Disables entering input into the spin field.

© Copyright IBM Corp. 1994

rightAlign
Right-justifies the text or number.

servant
Allows you to create a multiple-field spin button by spinning servants from the master. In order to use this style, you must call IBaseSpinButton::setMaster (see page 35) to define the master spin field.

Portability Considerations: The AIX release does not support the centerAlign and rightAlign styles. Spin button text and numbers always are left aligned in AIX.

Public Members

Figure 7. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 8. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 10. IBitFlag

Derivation

IBase
  IBitFlag

Header File: ibitflag.hpp

The IBitFlag class is the abstract base class for the bitwise styles and attributes used by window and control classes in the User Interface Class Library. Because this class in an abstract base class, you cannot create objects of this class.

Deriving Classes from IBitFlag

Typically, you can declare classes derived from IBitFlag by using the macros that accompany this class. Optionally, these macros lets you:

- Construct objects of one derived class from objects of another class derived from IBitFlag.
- Combine objects of one derived class with objects of another class derived from IBitFlag. For example, using the bitwise OR operator.

Macro Descriptions

You can use the following macros to declare classes derived from IBitFlag:

INESTEDBITFLAGCLASSDEF0 macro
Declares a logical base bitwise flag class scoped to another class. A logical base bitwise class is a class of bitwise flag objects that cannot be constructed from a bitwise flag object of another class.

INESTEDBITFLAGCLASSDEF1 macro
Declares a bitwise flag class whose objects can be constructed from an object of another bitwise flag class. The library assumes both bitwise flag classes have the same name and are scoped to another class.

INESTEDBITFLAGCLASSDEF2 macro
Declares a bitwise flag class whose objects can be constructed from an object of two other bitwise flag classes. The library assumes all the bitwise flag classes have the same name and are scoped to another class.

INESTEDBITFLAGCLASSDEF3 macro
Declares a bitwise flag class whose objects can be constructed from an object of three other bitwise flag classes. The library assumes all the bitwise flag classes have the same name and are scoped to another class.

INESTEDBITFLAGCLASSDEF4 macro
Declares a bitwise flag class whose objects can be constructed from an object of four other bitwise flag classes. The library assumes all the bitwise flag classes have the same name and are scoped to another class.

© Copyright IBM Corp. 1994

INESTEDBITFLAGCLASSFUNCS macro

Declares global functions that operate on a class of bitwise flag objects scoped to another class. The functions are global, rather than member functions, to allow for commutative operations between objects of different bitwise flag classes.

Note: Do not use this macro to define global functions for a logical base style class (one declared with the INESTEDBITFLAGCLASSDEF0 macro).

Constructors

```
protected:
IBitFlag(unsigned long value);
```

You can only construct objects of this class from an unsigned long value.

Note: This constructor is protected because objects derived from this class should not be arbitrarily constructed. To provide type safety for window and control constructors, you can only specify:

- Existing style objects
- Existing attribute objects
- Combinations of these objects

Public Members

asUnsignedLong

```
unsigned long asUnsignedLong() const;
```

Converts the object to an unsigned long value.

operator !=

```
Boolean operator !=(const IBitFlag &rhs) const;
```

Used to compare two bitflag values for inequality.

operator ==

```
Boolean operator ==(const IBitFlag &rhs) const;
```

Used to compare two bitflag values for equality.

Figure 9. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Protected Members

setValue

```
IBitFlag &setValue(unsigned long value);
```

Assigns an unsigned long value to the object.

Chapter 11. IBitmapControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IStaticText
              IBitmapControl

Header File: ibmpctl.hpp

Objects of the IBitmapControl class create and manage bit-map controls. A bit-map control draws a bit map for its background.

Motif Information: AIX does not support the ISystemBitmapHandle class. Therefore, you cannot use the constructor that takes an ISystemBitmapHandle object as a parameter.

Nested Classes

IBitmapControl::Style

Constructors

```
public:
IBitmapControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IResourceId &bitmapId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IBitmapControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   unsigned long bitmapId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IBitmapControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IBitmapHandle &bitmapId = IBitmapHandle ( ),
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

© Copyright IBM Corp. 1994

Condition: (2) OS/2 Only

```
public:
IBitmapControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   ISystemBitmapHandle::Identifier bitmapId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IBitmapControl(unsigned long id, IWindow *parentDialog);

public:
IBitmapControl(const IWindowHandle &handle);
```

You can construct objects of this class from the following:

- A nonsystem bit map. This creates a bit-map control with the specified window ID, parent and owner windows, screen position and size, bit map resource ID and resource library, and window style.
- A nonsystem bit map. This creates a bit-map control with the specified window ID, parent and owner windows, screen position and size, bit map resource ID and resource library, and window style.
- A nonsystem bit map. This creates a bit-map control with the specified window ID, parent and owner windows, screen position and size, bit map hande, and window style.
- A system bit map. This creates a bit-map control with the specified window ID, parent and owner windows, screen position and size, system bit map, and window style.
- A dialog template. This creates an IBitmapControl object for the specified bit-map dialog control.
- An existing bit-map control. This creates an IBitmapControl object for the specified bit-map control's window handle.

Public Members

bitmap

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

Returns the handle to the bit map.

defaultStyle

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

Returns the default style. This style is classDefaultStyle (see page 51) unless you have changed it using setDefaultStyle (see page 47).

moveSizeTo

Condition: (3) AIX only

```
virtual IBitmapControl &moveSizeTo(
                          const IRectangle &rectangle);
```

Changes the position and size of the window.

setBitmap

Condition: (2) OS/2 Only

```
virtual IBitmapControl &setBitmap(
                          ISystemBitmapHandle::Identifier bitmap);

virtual IBitmapControl &setBitmap(unsigned long bitmapId);

virtual IBitmapControl &setBitmap(
                          const IResourceId &bitmapId);

virtual IBitmapControl &setBitmap(
                          const IBitmapHandle &handle);
```

Changes a bit-map control window.

setDefaultStyle

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

Sets the default style for all subsequent bit-map controls.

Figure 10. Inherited Public Members

Member function	Class
activeColor	IWindow
alignment	IStaticText
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IStaticText
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableFillBackground	IStaticText

Figure 10. Inherited Public Members

Member function	Class
disableGroup	IControl
disableHalftone	IStaticText
disableNotification	INotifier
disableNotification	IWindow
disableStrikeout	IStaticText
disableTabStop	IControl
disableUnderscore	IStaticText
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableFillBackground	IStaticText
enableGroup	IControl
enableHalftone	IStaticText
enableNotification	INotifier
enableNotification	IWindow
enableStrikeout	IStaticText
enableTabStop	IControl
enableUnderscore	IStaticText

Figure 10. Inherited Public Members

Member function	Class
enableUpdate	IWindow
fillColor	IStaticText
font	IWindow
foregroundColor	IWindow
foregroundColor	IStaticText
handle	IWindow
handleException	IWindow
hasFillBackground	IStaticText
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHalftone	IStaticText
isLayoutDistorted	IWindow
isShowing	IWindow
isStrikeout	IStaticText
isTabStop	IWindow
isTabStop	IControl
isUnderscore	IStaticText
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
limit	IStaticText
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow

Figure 10. Inherited Public Members

Member function	Class
moveSizeTo	IStaticText
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetFillColor	IStaticText
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IStaticText
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFillColor	IStaticText
setFocus	IWindow

Figure 10. Inherited Public Members

Member function	Class
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setLimit	IStaticText
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IStaticText
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size this bit-map control can be based on the actual size of the currently set bit map.

Figure 11. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow

Figure 11. Inherited Protected Members

Member function	Class
calcLimitSize	IStaticText
calcMinimumSize	IWindow
calcMinimumSize	IStaticText

Figure 11. Inherited Protected Members

Member function	Class
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	IStaticText
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 12. IBitmapControl::Style

Derivation

IBase
  IBitFlag
    IBitmapControl::Style

Header File: ibmpctl.hpp

The nested class IBitmapControl::Style provides a set of valid bit-map control styles for the static member functions of the class IBitmapControl (see page 45). You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IStatic::Style (see page 846)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

sizeToBitmap
Sizes the window to the size of the currently set bit map.

Public Members

Figure 12. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

© Copyright IBM Corp. 1994

Figure 13. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 13. IBitmapHandle

Derivation

IBase
  IHandle
    IBitmapHandle

Header File: ihandle.hpp

Objects of the IBitmapHandle class access and manage bit-map resources. IBitmapHandle objects manage bit-map resources through reference counting. Reference counting allows the system to use one bit map in multiple places, and the library maintains the lifetime of this bit map until all users are finished with it.

Motif Information: IBitmapHandle is an alias for the XLib Pixmap type.

Presentation Manager Information: IBitmapHandle is an alias for the OS/2 Programmer's Toolkit type HBITMAP.

Constructors

```
public:
IBitmapHandle(Value value = 0);

public:
IBitmapHandle(const IBitmapHandle &aHandle);
```

You can construct objects of this class from either a bit-map handle (a value of type IHandle::Value) or an existing IBitmapHandle object.

The library provides the copy constructor, destructor, and assignment operator to keep track of references to the bit map so its resources can be freed when no longer needed.

Public Members

operator =

```
IBitmapHandle &operator =(const IBitmapHandle &handle);
```

Assigns the value of one bit-map handle to another while managing the references to the two handles. If the reference count of the bit map previously associated with the target handle reaches 0, the bit map is destroyed.

© Copyright IBM Corp. 1994

operator Value

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

Returns the IBitmapHandle value.

Figure 14. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Chapter 14. IButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton

Header File: ibutton.hpp

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.

Nested Classes

IButton::Style

Constructors

```
protected:
IButton();
```

The only way to create objects of this class is from a derived class. To enforce this, the only constructors we provide for this class are protected.

Public Members

allowsMouseClickFocus

Condition: (1) Ignored on AIX

```
Boolean allowsMouseClickFocus() const;
```

Returns whether the button can receive the focus when clicked with the mouse.

Motif Information: This function always returns true.

backgroundColor

Condition: (2) OS/2 Only

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

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. 1994

click

```
virtual IButton &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.

disabledForegroundColor

Condition: (2) OS/2 Only

```
virtual IColor disabledForegroundColor() const;
```

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.

disableMouseClickFocus

Condition: (1) Ignored on AIX

```
virtual IButton &disableMouseClickFocus();
```

Prevents the button from receiving the focus when the user selects it using the mouse.

enableMouseClickFocus

Condition: (1) Ignored on AIX

```
virtual IButton &enableMouseClickFocus(
                   Boolean enable = true);
```

Enables the button to receive the focus when the user selects it using the mouse.

enable Flag indicating whether the mouse can have focus.

Motif Information: In the AIX version, buttons always receive the focus when clicked on.

foregroundColor

Condition: (2) OS/2 Only

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

Returns the foreground color value of the window area, or the default if no color for the area has been set.

highlight

Condition: (1) Ignored on AIX

```
virtual IButton &highlight(Boolean highlight = true);
```

Sets the button's highlight state. A highlighted button looks the same as a button that has the mouse selection button pressed while the mouse pointer is over the button control.

Motif Information: This function always returns false.

hiliteBackgroundColor

Condition: (2) OS/2 Only

```
virtual IColor hiliteBackgroundColor() const;
```

Returns the highlight background color value of the window area, or the default if no color for the area has been set.

hiliteForegroundColor

Condition: (2) OS/2 Only

```
virtual IColor hiliteForegroundColor() const;
```

Returns the highlight foreground color value of the window area, or the default if no color for the area has been set.

isHighlighted

Condition: (1) Ignored on AIX

```
Boolean isHighlighted() const;
```

If the button's highlight state is set, true is returned. A highlighted button looks the same as a button that has the mouse selection button pressed while the mouse pointer is over the button control.

Motif Information: This function always returns false.

setText

```
virtual IButton &setText(const IResourceId &text);

virtual IButton &setText(const char *text);
```

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.

unhighlight

Condition: (1) Ignored on AIX

```
virtual IButton &unhighlight();
```

Turns off the button's highlight state. A highlighted button looks the same as a button that has the mouse selection button pressed while the mouse pointer is over the button control.

Motif Information: This function always returns false.

Figure 15. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow

Figure 15. Inherited Public Members

Member function	Class
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow

Figure 15. Inherited Public Members

Member function	Class
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 16. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 15. IButton::Style

Derivation

IBase
  IBitFlag
    IButton::Style

Header File: ibutton.hpp

The nested class IButton::Style provides a valid button style for the defaultStyle and setDefaultStyle functions and the IButton constructor. You can use this style with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

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.

Motif Information: AIX does not support the noPointerFocus style.

Public Members

Figure 17. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 18. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 16. ICanvas

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ICanvas
```

Header File: icanvas.hpp

Objects of the ICanvas class are client window objects. A client window object is an intermediate window between an IFrameWindow and other child windows. To specify the client window for an IFrameWindow, use IFrameWindow::setClient (see page 363).

Unlike other canvas classes, ICanvas does not alter the size or position of its child windows. However, it does provide additional controls that are missing if you use another class (such as IStaticText) or no window at all.

When you use ICanvas as a client window or as the parent window of controls that can accept the input focus, ICanvas provides the following controls:

- Handles tab and cursor movement keys
- Handles focus change events
- Allows the client window or the frame window (if a canvas is not the client window) to be the control point for command processing
- Allows the frame window to change the appearance of the mouse pointer
- Prevents the border of the frame window from being overwritten

Additionally, this class provides the above behavior for all derived classes. Derived classes can provide special layout rules for sizing and positioning their child windows.

Note: When you add an IListBox as a child window of an ISplitCanvas object, you must give it the IListBox::noAdjustPosition (see page 477) style.

Nested Classes

ICanvas::Style

Constructors

```
public:
ICanvas(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initial = IRectangle ( ),
```

© Copyright IBM Corp. 1994

```
   const Style &style = defaultStyle ( ));

protected:
ICanvas();
```

You can construct objects of this class with the following parameters:

windowId Window identifier of the canvas you are constructing.

Note: For OS/2, this value should be limited to the range 0 through 65535.

It is recommended that you:

- Give unique identifiers to all windows in the same frame window. "In the same frame window" means that the frame window is the first frame in its parent window chain.
- Give the client window a window ID of FID_CLIENT (0x8008).
- Do not use the other FID_xxx values.

parent The parent window of the canvas you are constructing. You must specify a parent window. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0.

The parent window is primarily used for visible relationships.

owner Owner window of the canvas you are constructing.

Note: In OS/2, the owner window is primarily used for routing notification events and unprocessed messages. OS/2 also uses the owner window chain to inherit colors.

In Motif, the owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial (Optional) The initial position and size of the canvas you are constructing. The initial position is the lower-left corner of the canvas relative to the lower-left corner of the parent window.

style (Optional) Specifies the window's characteristics using ICanvas::Style (see page 70).

For Motif applications, you can also use the protected ICanvas constructor for derived classes that do not need the BulletinBoard widget that the above constructor creates.

Public Members

backgroundColor

Condition: (2) OS/2 Only

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

Returns the background color value of the window area. The default color is returned if no color has been set.

defaultPushButton

```
virtual IWindowHandle defaultPushButton() const;
```

Returns the current push button with focus.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 70) unless you have changed the style using setDefaultStyle (see page 65).

isTabStop

```
virtual Boolean isTabStop() const;
```

If the canvas can be tabbed to, true is returned. If any of the canvas's child windows can be tabbed to, the canvas can be tabbed to.

matchForMnemonic

```
virtual IWindowHandle matchForMnemonic(
                        unsigned short character) const;
```

Returns the first child window using the specified character as a mnemonic.

origDefaultButtonHandle

```
IWindowHandle origDefaultButtonHandle() const;
```

Returns the push button originally identified as the default.

setDefaultStyle

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

Sets the default style for all subsequent canvases.

style Use the styles provided by ICanvas::Styles (see page 70) or IWindow::Styles to specify the default style.

setLayoutDistorted

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

Provides notification to a canvas that its layout of child windows needs to be updated. All windows maintain an internal state and use setLayoutDistorted to update that state and to communicate any changes to the parent window.

The canvas object uses these state changes to update the size and position of its child windows. The internal state of a window is the enumeration IWindow::Layout (see page 953).

ICanvas::setLayoutDistorted adds the IWindow::layoutDistorted flag to its internal state if it receives a value of IWindow::childWindowCreated or IWindow::childWindowDestroyed and calls IWindow::setLayoutDistorted. If the IWindow::immediateUpdate flag is on, the canvas runs its layout routine.

Figure 19. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow

Figure 19. Inherited Public Members

Member function	Class
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow

Figure 19. Inherited Public Members

Member function	Class
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow

Figure 19. Inherited Public Members

Member function	Class
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

areChildrenReversed
```
Boolean areChildrenReversed() const;
```

States whether the order in which child windows are returned by the class IWindow::ChildCursor (see page 954) matches the order in which the child windows were constructed.

This returns false only if the static function IWindow::setDefaultOrdering is called with the enumerator IWindow::onTopOfSiblings and the canvas has not run its layout routine.

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

Returns the minimum screen size this control can occupy, based on the size and positions of its child windows.

fixupChildren
```
virtual IWindowPosBuffer fixupChildren();
```

Generates a list of all child windows, determines if any can be tabbed to, and allows for reversing the order in which they are iterated.

layout
```
virtual ICanvas &layout();
```

Derived classes use this function to size and position child windows.

layoutSize
```
ISize layoutSize() const;
```

Returns the size needed to show all child windows.

passEventToOwner
Condition: (3) AIX only

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

Determines whether an event can be passed to the owner of this control.

registerCallbacks
Condition: (3) AIX only

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

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.

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

setLayoutSize

```
ICanvas &setLayoutSize(const ISize &size);
```

Sets the size needed to show all child windows.

unregisterCallbacks

Condition: (3) AIX only

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

Removes callbacks and X events added by registerCallbacks.

Figure 20. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow

Figure 20. Inherited Protected Members

Member function	Class
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 17. ICanvas::Style

Derivation

IBase
  IBitFlag
    ICanvas::Style

Header File: icanvas.hpp

The nested class ICanvas::Style provides a set of valid canvas styles for the defaultStyle and setDefaultStyle functions, and for the constructor of the class ICanvas (see page 63). You can use these styles with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

classDefaultStyle Specifies the original default style for this class, which is IWindow::visible.

Public Members

Figure 21. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 22. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 18. ICheckBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              ISettingButton
                ICheckBox

Header File: icheckbx.hpp

Objects of the ICheckBox class create and operate a check box.

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.

You should process selection of a check box by deriving objects from the ISelectHandler (see page 764) class and adding a handler to either the check box object or to its owner window.

You can attach the following handlers to this control:

- .IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)
- ISelectlHandler (see page 764)

Nested Classes

ICheckBox::Style

Constructors

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

public:
ICheckBox(unsigned long id, IWindow *parentDialog);
```

© Copyright IBM Corp. 1994

```
public:
ICheckBox(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Creating the specified 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 (Optional) The initial position and size of the check box you are constructing. The default constructor is IRectangle (see page 717).

  style (Optional) The check box's characteristics.

- Creating the object for the specified owner window.

  parentDialog
  The owner window.

- Creating the object using the handle of a check box window.

  handle The window handle of an existing check box control.

Public Members

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 77) unless you have changed the style using setDefaultStyle (see page 73).

disableAutoSelect

```
virtual ICheckBox &disableAutoSelect();
```

Removes the style autoSelect (see page 77) from the check box control.

Motif Information: AIX does not support this function, therefore it has no effect.

enableAutoSelect

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

Sets the check box control to the style autoSelect (see page 77), or removes the autoSelect style from the control.

Motif Information: The AIX version does not support this function, therefore it has no effect. In the AIX version, check boxes are always autoselected.

isAutoSelect

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

If the check box control has the style autoSelect (see page 77) set, true is returned.

Motif Information: AIX does not support this function and it therefore returns true. In the AIX version, check boxes are always autoselected.

setDefaultStyle

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

Sets the default style for all subsequent check boxes.

style Use the styles provided by ICheckBox::Styles (see page 77) to specify the default style.

Figure 23. Inherited Public Members

Member function	Class
activeColor	IWindow
allowsMouseClickFocus	IButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IButton
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
click	IButton
defaultPushButton	IWindow
deselect	ISettingButton
disable	IWindow
disableAutoSelect	ISettingButton
disabledBackgroundColor	IWindow

Figure 23. Inherited Public Members

Member function	Class
disabledForegroundColor	IWindow
disabledForegroundColor	IButton
disableGroup	IControl
disableMouseClickFocus	IButton
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableAutoSelect	ISettingButton
enableGroup	IControl
enableMouseClickFocus	IButton
enableNotification	INotifier
enableNotification	IWindow
enableNotification	ISettingButton
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow

Figure 23. Inherited Public Members

Member function	Class
foregroundColor	IWindow
foregroundColor	IButton
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
highlight	IButton
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IButton
hiliteForegroundColor	IWindow
hiliteForegroundColor	IButton
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isAutoSelect	ISettingButton
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHighlighted	IButton
isLayoutDistorted	IWindow
isSelected	ISettingButton
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier

Figure 23. Inherited Public Members

Member function	Class
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
select	ISettingButton
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow

Figure 23. Inherited Public Members

Member function	Class
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IButton
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
unhighlight	IButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size this check box control can be based on the text string length and the current font.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

unregisterCallbacks

Condition: (3) AIX only

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

Unregisters Motif callbacks for the widget created during object construction of this class.

Figure 24. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ISettingButton
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 19. ICheckBox::Style

Derivation

IBase
  IBitFlag
    ICheckBox::Style

Header File: icheckbx.hpp

The nested class ICheckBox::Style provides a valid set of check box styles for the static member functions of the class ICheckBox (see page 71). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IButton::Style (see page 61)

Styles

autoSelect
Specifies whether the check box is automatically selected and deselected as the user clicks on the check box. Otherwise, the application is responsible for selecting the check box.

classDefaultStyle
Specifies the original default style for this class, which is autoSelect | IWindow::visible.

Motif Information: The AIX version is always autoSelect style. You cannot disable auto selection of check boxes or radio buttons.

Public Members

Figure 25. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 26. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 20. ICnrControlList

Derivation

ISet
  ICnrControlList

Header File: icnrclst.hpp

Objects of the ICnrControlList class define a collection of IContainerControl objects. The User Interface Class Library uses this class in the implementation of the class IContainerControl (see page 149).

ICnrControlList is derived from the ISet class of the IBM Collection Class Library.

Constructors

```
public:
ICnrControlList();

public:
ICnrControlList(const ICnrControlList &);
```

You can construct objects of this class from the following:

- The default constructor, which accepts no parameters
- A pre-existing ICnrControlList object

© Copyright IBM Corp. 1994

Chapter 21. ICnrEmphasisEvent

Derivation

```
IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrEmphasisEvent
```

Header File: icnrevt.hpp

Objects of the ICnrEmphasisEvent class are dispatched whenever the container record's attributes are changed. These events are created in response to a change-in-emphasis notification from the container. Emphasis is a visible indication of the condition of an object that can affect the ability of a user to interact with that object.

Constructors

```
public:
ICnrEmphasisEvent(const IControlEvent &evt);
```

Condition: (3) AIX only

```
public:
ICnrEmphasisEvent(
   IControlEvent &evt,
   IContainerObject *object,
   Boolean acquiring);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

changed

```
Boolean changed(IContainerObject::Emphasis emphasis) const;
```

If the specified emphasis value has changed, true is returned.

isAcquired

Condition: (3) AIX only

```
Boolean isAcquired() const;
```


© Copyright IBM Corp. 1994

Returns true if the ICnrEmphasisEvent is for an object that is acquiring emphasis. Otherwise, it returns false. The ICnrHandler::selectedChanged function uses this function.

object

```
IContainerObject *object() const;
```

Returns the object that has changed its emphasis.

Figure 27. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Protected Members

Figure 28. Inherited Protected Members

Member function	Class
containerId	ICnrEvent

Note: * Application Support Class Library

Chapter 22. ICnrEnterEvent

Derivation

```
IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrEnterEvent
```

Header File: icnrevt.hpp

Objects of the ICnrEnterEvent class are dispatched when an enter event occurs in the container. This can be as a result of the following:

- The **Enter** key
- A double click of the mouse select button

These events are created in response to an enter notification from the container.

Constructors

```
public:
ICnrEnterEvent(const IControlEvent &evt);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

enterPressed

```
Boolean enterPressed() const;
```

If the event is the result of pressing the keyboard **Enter** key, true is returned. Otherwise, the event occurred because the select button was double clicked.

object

```
IContainerObject *object() const;
```

Returns the object over which the enter occurred. If the enter did not occur on an object, 0 is returned.

© Copyright IBM Corp. 1994

validObject

```
Boolean validObject() const;
```

If the event occurred over an object, true is returned.

Figure 29. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Protected Members

Figure 30. Inherited Protected Members

Member function	Class
containerId	ICnrEvent

Note: * Application Support Class Library

Chapter 23. ICnrEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent

Header File: icnrevt.hpp

ICnrEvent is the base class for all container events classes.

Constructors

```
public:
ICnrEvent(const IControlEvent &evt);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

Figure 31. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent

Figure 31. Inherited Public Members

Member function	Class
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

containerId

```
unsigned long containerId() const;
```

Returns the control ID for the container.

Chapter 24. ICnrHandler

Derivation

IBase
  IVBase
    IHandler
      ICnrHandler

Header File: icnrhdr.hpp

Objects of the ICnrHandler class provide basic event handling capability for the container.

Create a handler derived from ICnrHandler and attach it to the container whose events are to be handled. You can do this by calling handleEventsFor (see page 88) to pass the container to the container handler.

ICnrHandler dispatches the following events in the container:

- Changing selection, in-use, or cursored emphasis for an object in the container. This is described in ICnrEmphasisEvent (see page 80).
- Scrolling past the container's delta value. This is described in ICnrQueryDeltaEvent (see page 97).
- Responding to an enter event using the keyboard or mouse. This is described in ICnrEnterEvent (see page 83).
- Requesting help in the container. This is described in ICnrHelpEvent (see page 91).
- Expanding and collapsing nodes in a tree view. This is described in IContainerObject (see page 209).
- Scrolling the container window. This is described in ICnrScrollEvent (see page 100).

When the container handler receives one of these events, it creates a corresponding event object and routes that object to the appropriate ICnrHandler virtual function. You must 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 event requires no additional processing. Do not pass it to another handler.

false If there is another handler for the container, pass the event to the next handler.

© Copyright IBM Corp. 1994

Constructors

```
protected:
ICnrHandler();
```

You can only construct objects of this class by using the default constructor, which does not accept any parameters.

Public Members

cursoredChanged

Condition: (1) Ignored on AIX

```
virtual Boolean cursoredChanged(ICnrEmphasisEvent &event);
```

Called when the cursored emphasis of an object changes. By default, this function calls IContainerObject::handleCursoredChange (see page 211) on the object whose emphasis has changed.

deltaReached

Condition: (1) Ignored on AIX

```
virtual Boolean deltaReached(ICnrQueryDeltaEvent &event);
```

Called when the container is scrolled to the predefined delta.

enter

```
virtual Boolean enter(ICnrEnterEvent &event);
```

Called when the user selects the **Enter** key or double-clicks the mouse select button on an object. If the enter occurs over an object, by default, this function calls IContainerObject::handleOpen (see page 212). Otherwise, this function ignores the enter.

handleEventsFor

```
virtual ICnrHandler &handleEventsFor(
                        IContainerControl *container);
```

Attaches the handler to the specified container control.

help

```
virtual Boolean help(ICnrHelpEvent &event);
```

Called when help is requested in a container.

Motif Information: By default, this function does nothing.

inuseChanged

Condition: (1) Ignored on AIX

```
virtual Boolean inuseChanged(ICnrEmphasisEvent &event);
```

Called when the in-use emphasis of an object changes. By default, this function calls IContainerObject::handleInuseChange (see page 211).

selectedChanged

```
virtual Boolean selectedChanged(ICnrEmphasisEvent &event);
```

Called when the selection emphasis of an object changes. By default, this function calls IContainerObject::handleSelectedChange (see page 212).

stopHandlingEventsFor

```
virtual ICnrHandler &stopHandlingEventsFor(
                        IContainerControl *container);
```

Detaches the handler from the specified container control.

treeCollapsed

```
virtual Boolean treeCollapsed(
                  IContainerObject *collapsedObject,
                  IContainerControl *container);
```

Called when the specified object is collapsed in the specified container. By default, this function calls IContainerObject::handleTreeCollapse (see page 212).

treeExpanded

```
virtual Boolean treeExpanded(
                  IContainerObject *expandedObject,
                  IContainerControl *container);
```

Called when the specified object is expanded in the specified container. By default, this function calls IContainerObject::handleTreeExpand (see page 212) on the expanded object.

windowScrolled

Condition: (1) Ignored on AIX

```
virtual Boolean windowScrolled(ICnrScrollEvent &event);
```

Called when the container window is scrolled.

Motif Information: By default, this function does nothing.

Figure 32. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a container event is received, the appropriate virtual function is called.

Figure 33. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 25. ICnrHelpEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrHelpEvent

Header File: icnrevt.hpp

Objects of the ICnrHelpEvent class are dispatched when a user requests help in the container. These events are created in response to a notification from the container that the user has requested help.

Constructors

```
public:
ICnrHelpEvent(const IControlEvent &chpevt);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

column

```
IContainerColumn *column() const;
```

If help was requested within a multiple-line edit field, the column being edited is returned. Otherwise, 0 is returned.

object

```
IContainerObject *object() const;
```

Returns the object over which help is requested. Otherwise, it returns 0.

Figure 34. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*

Figure 34. Inherited Public Members

Member function	Class
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent

© Copyright IBM Corp. 1994

Figure 34. Inherited Public Members

Member function	Class
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Protected Members

Figure 35. Inherited Protected Members

Member function	Class
containerId	ICnrEvent

Note: * Application Support Class Library

Chapter 26. ICnrMenuHandler

Derivation

IBase
  IVBase
    IHandler
      IMenuHandler
        ICnrMenuHandler

Header File: icnrmhdr.hpp

Objects of the ICnrMenuHandler class extend the class IMenuHandler (see page 508) by providing specific information related to the menu. Primarily, this means providing the object over which the menu was requested.

Create a handler derived from ICnrMenuHandler and attach it to the container whose menu events are to be handled. You can do this by calling handleEventsFor (see page 93) to pass the appropriate container to the container menu handler.

When the container menu handler receives a container menu event, it creates an IMenuEvent (see page 506) object and routes that object to the appropriate ICnrMenuHandler virtual function. You must override these virtual functions to supply your own specialized processing of a container menu event.

The return value from the virtual functions specifies whether the container menu event is passed on for additional processing, as follows:

true The container menu event requires no additional processing. Do not pass it to another handler.

false If there is another handler for the container, pass the container menu event to the next handler.

Constructors

```
protected:
ICnrMenuHandler();
```

You can only construct objects of this class by using the default constructor, which does not accept any parameters.

Public Members

handleEventsFor

```
virtual ICnrMenuHandler &handleEventsFor(
                            IContainerControl *container);
```

Attaches the handler to the specified container control.

© Copyright IBM Corp. 1994

Note: This handler cannot be attached to IWindow*, and the inherited function for doing so is private.

popupMenuObject

```
IContainerObject *popupMenuObject();
```

Returns the object for which a pop-up menu was requested. If a pop-up menu was not requested for an object, 0 is returned.

stopHandlingEventsFor

```
virtual ICnrMenuHandler &stopHandlingEventsFor(
                            IContainerControl *container);
```

Detaches the handler from the specified container control.

Figure 36. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
handleEventsFor	IMenuHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler
stopHandlingEventsFor	IMenuHandler

Note: * Application Support Class Library

Protected Members

addSourceEmphasis

Condition: (1) Ignored on AIX

```
virtual void addSourceEmphasis(const IMenuEvent &menuEvent);
```

Adds source emphasis by calling IContainerControl::showSourceEmphasis (see page 180).

Note: A different overload of the same function is called.

- If the object is not selected, source emphasis is displayed on the object.
- If the object is selected and other objects are also selected, source emphasis is displayed on the group of objects.
- If a pop-up menu is selected off of an object, source emphasis is displayed on the container itself.

dispatchHandlerEvent

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

If a container menu event is received, the appropriate virtual function is called.

menuEnded

```
virtual Boolean menuEnded(IMenuEvent &menuEvent);
```

Resets the pointer to the menu object that is stored when the menu is created, and calls IMenuHandler::menuEnded (see page 510) to complete the processing.

removeSourceEmphasis

Condition: (1) Ignored on AIX

```
virtual void removeSourceEmphasis(
              const IMenuEvent &menuEvent);
```

Removes source emphasis by calling IContainerControl::hideSourceEmphasis (see page 161).

This function reverses the action of addSourceEmphasis.

Note: A different overload of the same function is called.

Figure 37. Inherited Protected Members

Member function	Class
addSourceEmphasis	IMenuHandler
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler
dispatchHandlerEvent	IMenuHandler
makePopUpMenu	IMenuHandler
menuEnded	IMenuHandler
menuSelected	IMenuHandler
menuShowing	IMenuHandler
removeSourceEmphasis	IMenuHandler

Note: * Application Support Class Library

Chapter 27. ICnrObjectSet

Derivation

ISequence
  ICnrObjectSet

Header File: icnrolst.hpp

Objects of the ICnrObjectSet class define a set of IContainerObjects (see page 209). IContainerControl (see page 149) provides several functions that return sets of IContainerObjects.

ICnrObjectSet is derived from the ISequence class of the IBM Collection Class Library.

Constructors

```
public:
ICnrObjectSet();

public:
ICnrObjectSet(const ICnrObjectSet &);
```

You can construct objects of this class from the following:

- The default constructor, which accepts no parameters
- A pre-existing ICnrObjectSet object

© Copyright IBM Corp. 1994

Chapter 28. ICnrQueryDeltaEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrQueryDeltaEvent

Header File: icnrevt.hpp

Objects of the ICnrQueryDeltaEvent class are dispatched when the delta value is reached in the container. These events are created in response to a notification from the container that the pre-set delta value has been reached.

Constructors

Condition: (1) Ignored on AIX

```
public:
ICnrQueryDeltaEvent(const IControlEvent &evt);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

atBottomDelta

Condition: (1) Ignored on AIX

```
Boolean atBottomDelta() const;
```

If the object that represents the delta value scrolls into view at the bottom of the container, true is returned.

atFirstObject

Condition: (1) Ignored on AIX

```
Boolean atFirstObject() const;
```

If the container has scrolled to the first object in the container, true is returned.

© Copyright IBM Corp. 1994

atLastObject

Condition: (1) Ignored on AIX

```
Boolean atLastObject() const;
```

If the container has scrolled to the last object in the container, true is returned.

atTopDelta

Condition: (1) Ignored on AIX

```
Boolean atTopDelta() const;
```

If the object that represents the delta value scrolls into view at the top of the client area, true is returned.

Figure 38. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Protected Members

Figure 39. Inherited Protected Members

Member function	Class
containerId	ICnrEvent

Note: * Application Support Class Library

Chapter 29. ICnrScrollEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrScrollEvent

Header File: icnrevt.hpp

Objects of the ICnrScrollEvent class are dispatched when scrolling is requested in the container. These events are created in response to a notification from the container that one of the container's windows has been scrolled.

Constructors

Condition: (1) Ignored on AIX

```
public:
ICnrScrollEvent(const IControlEvent &chpevt);
```

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (see page 90) creates objects of this class from an object of the class IControlEvent (see page 227) in reponse to the user pressing the **Enter** key or double-clicking the mouse select button.

Public Members

amount

Condition: (1) Ignored on AIX

```
long amount() const;
```

Returns the amount the window has scrolled, in pixels.

isHorizontal

Condition: (1) Ignored on AIX

```
Boolean isHorizontal() const;
```

If scrolling in the horizontal direction, true is returned.

© Copyright IBM Corp. 1994

isLeftDetails

Condition: (1) Ignored on AIX

```
Boolean isLeftDetails() const;
```

If the scrolling occurred in the left (or only) side of the details view, true is returned.

isRightDetails

Condition: (1) Ignored on AIX

```
Boolean isRightDetails() const;
```

If the scrolling occurred in the right side of a split details view, true is returned.

isVertical

Condition: (1) Ignored on AIX

```
Boolean isVertical() const;
```

If scrolling occurred in the vertical direction, true is returned.

Figure 40. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent

Figure 40. Inherited Public Members

Member function	Class
window	IEvent

Note: * Application Support Class Library

Protected Members

Figure 41. Inherited Protected Members

Member function	Class
containerId	ICnrEvent

Note: * Application Support Class Library

Chapter 30. IColor

Derivation

IBase
  IVBase
    IColor

Header File: icolor.hpp

Use the IColor class to query and set the color of various areas of the User Interface Class Library controls. This class is the base for the other color classes.

Motif Information: 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 recognize values from 0 to 65535.

Use the asPixel member function and the constructor that takes a pixel.

Constructors

```
public:
IColor(Color value);

public:
IColor(
   unsigned char red,
   unsigned char green,
   unsigned char blue);

public:
IColor(const IColor &color);

public:
IColor(long index);
```

Condition: (3) AIX only

```
public:
IColor(unsigned long pixel);

protected:
IColor();
```

You can construct objects of this class from:

- The default constructor (protected). The resulting object represents black because the red, green, and blue components are all set to 0.
- The enumeration Color (see page 106). This constructor uses a specified color enumerator to set the red, green, and blue components.

© Copyright IBM Corp. 1994

- Separate red, green, and blue values. The constructor stores the separate component values.
- The copy constructor. Use this constructor to copy the red, green, and blue values from the copied object.

Public Members

asPixel

Condition: (3) AIX only

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

Returns the default color-map index closest to the red, green and blue values.

asRGBLong

```
long asRGBLong() const;
```

Returns the red, green, and blue color value as a long.

Motif Information: 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).

blueMix

```
unsigned char blueMix() const;
```

Returns the value of the blue component.

greenMix

```
unsigned char greenMix() const;
```

Returns the value of the green component.

index

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

Returns the logical color table index closest to the red, green, and blue values.

Motif Information: The following list contains the index values and colors found in a logical color table. The values in parenthesis 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)
15	Pale gray (light gray)

Presentation Manager Information: 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

operator !=

```
Boolean operator !=(const IColor &color) const;
```

Returns true if the colors are not identical.

operator =

```
IColor &operator =(const IColor &color);
```

Replaces the color.

operator ==
```
Boolean operator ==(const IColor &color) const;
```

Returns true if the colors are identical.

redMix
```
unsigned char redMix() const;
```

Returns the value of the red component.

setBlue
```
IColor &setBlue(unsigned char blueMix);
```

Sets the blue color mix.

setGreen
```
IColor &setGreen(unsigned char greenMix);
```

Sets the green color mix.

setRed
```
IColor &setRed(unsigned char redMix);
```

Sets the red color mix.

value
```
Color value() const;
```

Using a specified color index, determines the Color enumerator.

Figure 42. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

Color

```
public:
enum Color
  {
  white,
  blue,
  red,
  pink,
  green,
  cyan,
  yellow,
  black,
  darkGray,
  darkBlue,
  darkRed,
  darkPink,
  darkGreen,
  darkCyan,
  brown,
  paleGray,
  };
```

Use the preceding enumerator values to specify colors.

Chapter 31. IComboBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IEntryField
              IComboBox

Header File: icombobx.hpp

Objects of the IComboBox class create and manage combination box control windows. The IComboBox class combines an entry field and a list box to form one control containing the features of both.

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

- IEditHandler (see page 247)
- IEditVerifyHandler (see page 253)
- IFocusHandler (see page 307)
- IResizeHandler (see page 732)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- ISelectHandler (see page 764)
- IShowListHandler (see page 788)

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

In OS/2, you must attach an IKeyboardHandler (see page 455) or IEditHandler (see page 247) to the entry field child of the combination box object in order to process keystrokes. You must wrapper the entry field using one of the IEntryField constructors.

In AIX, you can attach an IKeyboardHandler or IEditHandler directly to the IComboBox object. You should not wrapper the entry field within the combination box. You might want to use the IEditVerifyHandler (see page 253) instead of IKeyboardHandler for processing entry field changes.

Motif Information: The IComboBox constructor creates objects of this class using several Motif widgets. An XmForm widget is created with XmText, XmScrolledWindow, and XmList children. If the IComboBox object has the dropDownType or the dropDownListType style, an XmArrowButton is also created as a child of the XmForm widget. IWindow::handle (see page 928) returns the handle of the XmText widget.

© Copyright IBM Corp. 1994

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

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

Presentation Manager Information: The parent of an IComboBox should not set the IWindow::clipChildren style. This style prevents OS/2 from painting a small region below the list box portion of the IComboBox object.

Nested Classes

IComboBox::Cursor
IComboBox::Style

Constructors

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

Condition: (2) OS/2 Only

```
public:
IComboBox(unsigned long id, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
IComboBox(const IWindowHandle &handle);
```

You can construct objects of this class using the following:

- The ID, parent, owner, size, position, and style parameters
- The parent window and a combination-box ID
- The handle of an existing combination-box window

id A combination-box control ID.

parent The parent window.

owner The owner window.

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

style (Optional) The combination box's characteristics.

handle The window handle of an existing combination-box control.

Public Members

add

```
virtual unsigned long add(
                        unsigned long index,
                        const IResourceId &item);

virtual unsigned long add(
                        unsigned long index,
                        const char *text);

virtual unsigned long add(
                        unsigned long index,
                        const char * const *itemList,
                        unsigned long count = 1);

virtual IComboBox &add(const char *text, Cursor &cursor);

virtual IComboBox &add(
                      const IResourceId &item,
                      Cursor &cursor);
```

Inserts text into the list box portion of the combination box by either:

- Inserting the text in the list box portion and returning the index of the item or the first item selected
- Inserting the text at the cursor position and setting the cursor on the inserted 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.

item Resource ID of the text you want to insert in the list.

itemList An array of character strings.

count Number of text items in itemList. The default is 1 item.

addAscending

```
virtual unsigned long addAscending(const IResourceId &item);

virtual unsigned long addAscending(const char *text);
```

Inserts the specified item in ascending sort order and returns the index of the item inserted.

addAsFirst

```
virtual IComboBox &addAsFirst(
                      const char *text,
                      Cursor &cursor);

virtual unsigned long addAsFirst(const char *text);

virtual unsigned long addAsFirst(const IResourceId &item);

virtual IComboBox &addAsFirst(
                      const IResourceId &item,
                      Cursor &cursor);
```

Inserts text as the first item in the list box portion of the combination box by either:

- Inserting the text as the first item in the list box.
- Inserting the text and putting the cursor on the inserted item.

addAsLast

```
virtual IComboBox &addAsLast(
                      const char *text,
                      Cursor &cursor);

virtual unsigned long addAsLast(const char *text);

virtual unsigned long addAsLast(const IResourceId &item);

virtual IComboBox &addAsLast(
                      const IResourceId &item,
                      Cursor &cursor);
```

Inserts a line of text at the end of the list box portion of the combination box by either:

- Inserting the text as the last item in the list box.
- Inserting the text and putting the cursor on that item.

addAsNext

```
virtual IComboBox &addAsNext(
                      const char *text,
                      Cursor &cursor);

virtual IComboBox &addAsNext(
                      const IResourceId &item,
                      Cursor &cursor);
```

Inserts a line of text after the current cursor position in the list box portion of the combination box and places the cursor on that item.

addDescending

```
virtual unsigned long addDescending(
                        const IResourceId &item);

virtual unsigned long addDescending(const char *text);
```

Inserts the line of text in descending sort order and returns the index of the item inserted.

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

Returns the number of items in the list box.

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

Returns the default style. The default style is classDefaultStyle (see page 126) unless you have changed it using setDefaultStyle (see page 116).

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

Removes the selection state from the specified item.

index 0-indexed reference to item.

deselectAll
```
virtual IComboBox &deselectAll();
```

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

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

Returns the item's string at the cursor position.

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

Implements the public INotifier protocol for IComboBox classes. This member function causes the combination box to send notifications to any observer objects added.

hideList
```
virtual IComboBox &hideList();
```

Hides the list box.

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

If the list box is empty, true is returned.

isHorizontalScroll

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

If the list box has a horizontal scroll, true is returned.

isListShowing

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

If the list box is visible, true is returned.

isSelected

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

Returns true if the specified item is currently selected.

index Index of the item in a list box.

itemHandle

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

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.

itemText

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

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.

layoutAdjustment

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

Returns the dimensions that a window should be moved or sized 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.

locateText

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

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 (see page 122) to specify the type of search to perform. The default is exactMatch.

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

minimumRows

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

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

moveSizeTo

Condition: (3) AIX only

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

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.

numberOfSelections

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

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

position

Condition: (3) AIX only

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

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

remove

```
virtual unsigned long remove(unsigned long index);
```

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.

removeAll

```
virtual IComboBox &removeAll();
```

Removes all items from the list box.

removeAt

```
virtual IComboBox &removeAt(Cursor &cursor);
```

Removes the item at the cursor position and sets the cursor to the item following the removed item.

replaceAt

```
virtual IComboBox &replaceAt(
                    const char *text,
                    const Cursor &cursor);

virtual IComboBox &replaceAt(
                    const IResourceId &item,
                    const Cursor &cursor);
```

Replaces the specified item's string at the cursor position.

select

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

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.

selection

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

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

setBackgroundColor

Condition: (3) AIX only

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

Sets the background color to the specified color.

setDefaultStyle

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

Sets the default style for all subsequent combination boxes.

style Use the styles provided by IComboBox::Styles (see page 126) to specify the default style.

setForegroundColor

Condition: (3) AIX only

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

Sets the foreground color to the specified color.

setItemHandle

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

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.

setItemText

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

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

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.

setMinimumRows

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

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.

mimimumRows
Minimum number of visible rows.

setTop

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

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

index Index of the item in a list box.

showList

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

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

size

Condition: (3) AIX only

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

Returns the current size of the combination box.

top

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

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 IInvalidRequest exception is thrown.

topHandle

Condition: (3) AIX only

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

Returns the top-most manager widget of the combination box.

type

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

Returns the ControlType (see page 122) enumerator for the type of combination box.

visibleRectangle

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

Returns the window rectangle that is painted by the control.

Figure 43. Inherited Public Members

Member function	Class
activeColor	IWindow
alignment	IEntryField
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IEntryField
borderColor	IWindow
characterSize	IWindow
charType	IEntryField
childAt	IWindow
clear	IEntryField
copy	IEntryField
cut	IEntryField
defaultPushButton	IWindow
disable	IWindow
disable	IEntryField
disableAutoScroll	IEntryField
disableAutoTab	IEntryField
disableCommand	IEntryField
disableDataUpdate	IEntryField
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableInsertMode	IEntryField
disableMargin	IEntryField
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
discard	IEntryField
displaySize	ITextControl
enable	IWindow
enable	IEntryField
enableAutoScroll	IEntryField
enableAutoTab	IEntryField
enableCommand	IEntryField

Figure 43. Inherited Public Members

Member function	Class
enableDataUpdate	IEntryField
enableGroup	IControl
enableInsertMode	IEntryField
enableMargin	IEntryField
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IEntryField
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IEntryField
handle	IWindow
handleException	IWindow
hasChanged	IEntryField
hasFocus	IWindow
hasSelectedText	IEntryField
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isAutoScroll	IEntryField
isAutoTab	IEntryField
isCommand	IEntryField
isDisabled	IWindow
isEmpty	IEntryField
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isInsertMode	IEntryField
isLayoutDistorted	IWindow
isMargin	IEntryField
isReadOnly	IEntryField

Figure 43. Inherited Public Members

Member function	Class
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
leftIndex	IEntryField
limit	IEntryField
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveSizeTo	IEntryField
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
paste	IEntryField
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeAll	IEntryField
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow

Figure 43. Inherited Public Members

Member function	Class
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
selectedRange	IEntryField
selectedText	IEntryField
selectRange	IEntryField
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IEntryField
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setChangedFlag	IEntryField
setCharType	IEntryField
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setLeftIndex	IEntryField
setLimit	IEntryField
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl

Figure 43. Inherited Public Members

Member function	Class
text	IEntryField
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size of the combination box.

The size dimensions are:

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.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

setLayoutDistorted

Condition: (3) AIX only

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

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

unregisterCallbacks

Condition: (3) AIX only

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

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

Figure 44. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
calcMinimumSize	IEntryField
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
initialize	IEntryField
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	IEntryField
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IEntryField
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setLayoutDistorted	IEntryField
setNotificationHandler	IWindow
setStyle	IWindow
setStyle	IEntryField
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IEntryField
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

ControlType

```
public:
enum 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 display.

dropDownList
Specifies the drop-down list variation of the dropDown combination box. This drop-down list does not let a user type information into the entry field.

SearchType

```
public:
enum SearchType
  {
  prefix,
  substring,
  exactMatch,
  };
```

Use these enumerators to specify the type of search:

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.

Chapter 32. IComboBox::Cursor

Derivation

IBase
  IVBase
    IComboBox::Cursor

Header File: icombobx.hpp

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

Constructors

```
public:
Cursor(
   const IComboBox &comboBox,
   Filter type = selectedItems);
```

You can create objects of this class using either:

comboBox A combination-box control.

type Use the enumeration Filter (see page 125) to specify the type of filter for the combination box. The default is selectedItems.

Public Members

asIndex

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

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

invalidate

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

Flags the cursor as not valid.

isValid

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

Returns a Boolean depending on whether the cursor is pointing to a valid item. True means the cursor is pointing to a valid item.

© Copyright IBM Corp. 1994

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

Returns true if the cursor is set to the first list box item. If the cursor is not set successfully, false is returned.

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

Returns true if the cursor is set to the specified list box item. If the cursor is not set successfully, false is returned.

index Index of the item in the list box.

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

Returns true if the cursor is set to the last list box item. If the cursor is not set successfully, false is returned.

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

Returns true if the cursor is set to the next list box item. If no more items exist, the cursor is not valid and false is returned.

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

Returns true if the cursor is set to the previous list box item. If no previous item exist, the cursor is not valid and false is returned.

Figure 45. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

Filter

```
public:
enum 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.

Chapter 33. IComboBox::Style

Derivation

IBase
  IBitFlag
    IComboBox::Style

Header File: icombobx.hpp

The nested class IComboBox::Style provides a set of valid combination box styles for objects of the class IComboBox (see page 108). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

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 is to be converted from an ASCII code page into an EBCDIC code page, this style causes an entry field to not account for shift-in and shift-out characters introduced into its text.

This style is the opposite of mixedData.

classDefaultStyle
Specifies the original default style for this class, which is simpleType | IEntryField::leftAlign | anyData | IWindow::visible.

dbcsData
Sets the combination-box entry field to accept double-byte text only.

dropDownListType
Specifies the drop-down list variation of the drop-down combination box with an additional restriction that does not let a user type information into the entry field.

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

horizontalScroll
Provides horizontal scrolling for the list box control.

© Copyright IBM Corp. 1994

mixedData
Sets the combination box 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 an increase in the length of the data because of the addition of shift-in or shift-out characters, but does not exceed the text limit of the entry field.

sbcsData
Sets the combination-box entry field to accept single-byte text only.

simpleType
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.

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

Presentation Manager Information: 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 (see page 271).

Public Members

Figure 46. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 47. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 34. ICommandEvent

Derivation

IBase
  IVBase
    IEvent
      ICommandEvent

Header File: icmdevt.hpp

Objects of the ICommandEvent class represent an application-specific or system command. ICommandHandler (see page 131) constructs and processes objects of this class, typically as a result of the user selecting:

- 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

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.

Portability Considerations: In OS/2, ICommandEvents from accelerators go directly to the associated IFrameWindow. For portability, you should attach the ICommandHandler to the IFrameWindow which:

- 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

Presentation Manager Information: In Presentation Manager, the event result for WM_COMMAND and WM_SYSCOMMAND is reserved and the default window procedure does not pass either message up the owner or parent chain. Therefore, your callback functions do not have to set any result before they return true.

© Copyright IBM Corp. 1994

Constructors

```
public:
ICommandEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHandler::dispatchHandlerEvent (see page 404) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

commandId

```
unsigned long commandId() const;
```

Returns the ID of the push button or menu item, or 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.

isFromFrame

```
Boolean isFromFrame() const;
```

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.

source

```
Source source() const;
```

Returns the enumerator corresponding to the way in which the command event was initiated. The enumeration Source (see page 130) 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.

Figure 48. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent

Figure 48. Inherited Public Members

Member function	Class
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent

Figure 48. Inherited Public Members

Member function	Class
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Enumerations

Source

```
public:
enum Source
  {
  pushButton,
  menu,
  accelerator,
  unknown,
  };
```

These enumerators list the possible origin of the command event that ICommandEvent::source (see page 129) 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.

Chapter 35. ICommandHandler

Derivation

IBase
  IVBase
    IHandler
      ICommandHandler

Header File: icmdhdr.hpp

Objects of the ICommandHandler class process 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 (see page 403) to pass the appropriate window to the command handler.

When the command handler receives a command event, it creates an ICommandEvent (see page 128) 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 on to their owner window. A frame window passes application commands and the SC_CLOSE system command on to its client window.

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 the command event on for additional processing, as follows:

- If there is another handler for the window, pass the command event on to the next handler.
- If this is the last handler for the window, call IWindow::defaultProcedure (see page 946) to process the command event.

Note: Because IFrameHandler also processes ICommandEvents, you must think of it as if you had more than one ICommandHandler attached to a window.

© Copyright IBM Corp. 1994

Additionally, if you attach an IFrameHandler to the same window as an ICommandHandler, your ICommandHandler might not be called depending on the order that you attach these handlers.

Portability Considerations: In AIX, an accelerator sends events only directly down to their client windows, if they have a client window. Otherwise, the commands do not go anywhere else. In OS/2, 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 is used.

Motif Information: Motif has no concept of command events built into it. However, an event which 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 event. The library uses IWindowHandle::sendEvent to send the event to X. The control sends the event to its owner widget by definition, which matches Presentation Manager's system behavior for routing WM_COMMAND messages.

XtAddEventHandler is used to register interest in command events. This is done when ICommandHandler::handleEventsFor is invoked. Because ICommandEvents arrive at the owner of the push button, menu item, or accelerator, you ensure the ICommandHandler handles events for this owner window.

You can attach an ICommandHandler directly to a control. As in OS/2, the control must be its own owner. However, an owner window is also used for passing on unprocessed keyboard and mouse events. 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 same handler to more than one window
- More than one handler, of either the same or different types, to the same window

Constructors

```
public:
ICommandHandler();
```

You can only construct objects of this class by using the default constructor, which does not accept any parameters.

Public Members

Figure 49. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

command

```
virtual Boolean command(ICommandEvent &event);
```

Implemented by derived classes to handle application command events. You can use cmdEvent.window 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.

dispatchHandlerEvent

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

If a command event is received, the appropriate virtual function is called.

systemCommand

```
virtual Boolean systemCommand(ICommandEvent &event);
```

Implemented by derived classes to handle the system command events.

Figure 50. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 36. IContainerColumn

Derivation

IBase
  IVBase
    IContainerColumn

Header File: icnrcol.hpp

Objects of the IContainerColumn class provide the container with the ability to display a details view.

When using the details view, you must create an IContainerColumn object for each column in the details view. Use IContainerControl::addColumn to add the column objects to the container.

This class provides functions to add, delete, and hide columns in a container control. It also defines the characteristics of the column header (text or icon) and the position of the data in the column record.

The data in the object and the data displayed in the column are connected. For a user to have a details view, derive a class from IContainerObject (see page 209) and extend it with the additional fields to display in the columns of the container. You must ensure the column object has the following information:

- The exact offset of the data in the object
- The type of the data in the object

The column object must have the correct information because it handles the drawing of this data. If the column object does not have the correct information, the behavior of the column is undefined.

The default implementation of a container column has the following:

- A heading containing centered, read-only text
- A horizontal separator under the heading
- A vertical separator between columns
- String data that is non-editable, left-justified, and centered vertically

Note: You can add an IContainerColumn object to one container only.

Motif Information: The container widget automatically creates the leftmost column for details view. This column contains the icon and text of the corresponding IContainerObject. If you attempt to add columns to a container that represent the icon or text for an IContainerObject, the columns are ignored. However, if you add a column representing the text for an IContainerObject, the automatically created left-most column uses the headingText set for that column.

IContainerControl::ColumnCursor (see page 194) ignores the automatically created left-most column.

© Copyright IBM Corp. 1994

Constructors

```
public:
IContainerColumn(
   unsigned long dataOffset,
   const HeadingStyle &title = defaultHeadingStyle ( ),
   const DataStyle &data = defaultDataStyle ( ));

public:
IContainerColumn(
   DataSource objectDataType,
   const HeadingStyle &title = defaultHeadingStyle ( ),
   const DataStyle &data = defaultDataStyle ( ));

public:
IContainerColumn(const IContainerColumn &column);
```

You can construct objects of this class by:

- Providing the offset of the object data to be displayed in the column and, optionally, the styles to use for the heading and data.
- Providing a type of data that is part of IContainerObject (see page 209) (as opposed to the user portion of the object) and, optionally, the styles to use for the heading and data.
- Copying from an existing container column.

Public Members

dataAsDate

```
IDate dataAsDate(const IContainerObject *object) const;
```

Returns the data referenced by this column in the specified object. The data is returned as an IDate. If the type of the data is not IContainerColumn::date, an IInvalidRequest exception is thrown. You can call IContainerColumn::isDate to determine if this function is valid for a column object.

dataAsIcon

```
IPointerHandle dataAsIcon(
                const IContainerObject *object) const;
```

Returns the data referenced by this column in the specified object. The data is returned as an IPointerHandle (see page 657). If the type of the data is not IContainerColumn::icon, an IInvalidRequest exception is thrown. You can call IContainerColumn::isIconHandle to determine if this function is valid for a column object.

dataAsNumber

```
unsigned long dataAsNumber(
               const IContainerObject *object) const;
```

Returns the data referenced by this column in the specified object. The data is returned as an unsigned long integer. All data types can be returned as an unsigned long integer.

dataAsString

```
IString dataAsString(const IContainerObject *object) const;
```

Returns the data referenced by this column in the specified object. The data is returned as an IString. If the type of the data is not IContainerColumn::string, an IInvalidRequest exception is thrown. You can call IContainerColumn::isString to determine if this function is valid for a column object.

dataAsTime

```
ITime dataAsTime(const IContainerObject *object) const;
```

Returns the data referenced by this column in the specified object. The data is returned as an ITime. If the type of the data is not IContainerColumn::time (see page 135), an IInvalidRequest exception is thrown. You can call IContainerColumn::isTime to determine if this function is valid for a column object.

defaultDataStyle

```
static Style defaultDataStyle();
```

Returns the default data style. The default data style is classDefaultDataStyle unless you change it using setDefaultDataStyle.

defaultHeadingStyle

```
static Style defaultHeadingStyle();
```

Returns the default heading style. The default heading style is classDefaultHeadingStyle unless you have changed it using setDefaultHeadingStyle (see page 142).

disableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &disableDataUpdate();
```

Prevents editing of the column data.

disableHeadingUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &disableHeadingUpdate();
```

Prevents editing of the heading.

displayWidth

Condition: (1) Ignored on AIX

```
virtual unsigned long displayWidth();
```

Returns the displayable width of the column, in pixels.

If you want the column object's width, use IContainerControl::detailObjectRectangle instead of IContainerColumn::displayWidth. This is because a container details view cell is like a rectangle inside of another rectangle:

- The inner rectangle is the data rectangle of the cell and does not contain the required column margin.

  This rectangle's width is either:
  - The value set via IContainerColumn::setDisplayWidth
  - The value queried via IContainerColumn::displayWidth
- The outer rectangle is the complete cell, including the margins that the container control requires.

This is the value returned by IContainerControl::detailObjectRectangle.

Presentation Manager Information: This function no longer permits you to use private container memory. This only affects users of OS/2 2.0 who have not installed the latest Corrective Service Diskette (CSD). If you have installed OS/2 2.1 or OS/2 2.0 with the latest CSD, you should not be affected.

enableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &enableDataUpdate(
                            Boolean enable = true);
```

Permits an edit field to be opened in the column data. This is valid only if the style is string.

enableHeadingUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &enableHeadingUpdate(
                            Boolean enable = true);
```

Permits an edit field to be opened on the heading. This is valid only if the style is string.

hasHorizontalSeparator

```
Boolean hasHorizontalSeparator() const;
```

If the column heading has a separator drawn under it, true is returned.

hasVerticalSeparator

```
Boolean hasVerticalSeparator() const;
```

If the column heading has a vertical separator drawn after the column, true is returned.

headingIcon

Condition: (1) Ignored on AIX

```
virtual IPointerHandle headingIcon() const;
```

If the style is icon, the icon in the heading is returned. Otherwise an IPointerHandle with a value of 0 is returned.

headingText

```
virtual IString headingText() const;
```

If the style is string, the text in the heading is returned. Otherwise, an empty IString is returned.

helpId

```
unsigned long helpId() const;
```

Retrieves the help panel ID. You set the help panel ID by calling the function IContainerColumn::setHelpId. If you have attached an ICnrHandler to the container, the handler will display this help panel when the user requests help while direct-editing the column data of the details view.

hide

```
virtual IContainerColumn &hide();
```

Hides the column by making it invisible.

hideSeparators

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &hideSeparators(
                            const DataStyle &separatorStyles = horizontalSeparator ] verticalSeparator);
```

Removes either:

- The vertical separator after a column
- The horizontal separator under the column heading
- Both of the preceding separators

By default, this function removes both.

horizontalDataAlignment

Condition: (1) Ignored on AIX

```
HorizontalAlignment horizontalDataAlignment() const;
```

Returns the current horizontal alignment of the column data in a container details view. The returned value is an enumerator provided by HorizontalAlignment (see page 146).

horizontalHeadingAlignment

Condition: (1) Ignored on AIX

```
HorizontalAlignment horizontalHeadingAlignment() const;
```

Returns the current horizontal alignment of the column heading data in a container details view. The returned value is an enumerator provided by HorizontalAlignment (see page 146).

isDate

```
Boolean isDate() const;
```

If the data in the column is a date, true is returned.

isHeadingIconHandle

Condition: (1) Ignored on AIX

```
Boolean isHeadingIconHandle() const;
```

If the data in the column heading is an icon, true is returned.

isHeadingReadOnly

Condition: (1) Ignored on AIX

```
isHeadingReadOnly() const;
```

If the data in the column heading cannot be edited, true is returned.

Note: This function is obsolete. Use IContainerColumn::isHeadingWriteable instead.

isHeadingString

Condition: (1) Ignored on AIX

```
Boolean isHeadingString() const;
```

If the data in the column heading is a character string, true is returned.

isHeadingWriteable

Condition: (1) Ignored on AIX

```
Boolean isHeadingWriteable() const;
```

If the data in the column heading can be edited, true is returned.

isIconHandle

```
Boolean isIconHandle() const;
```

If the data in the column is an icon, true is returned.

isNumber

```
Boolean isNumber() const;
```

If the data in the column is a number, true is returned.

isReadOnly

Condition: (1) Ignored on AIX

```
isReadOnly() const;
```

If the data in column cannot be edited, true is returned.

Note: This function is obsolete. Use IContainerColumn::isWriteable instead.

isString

```
Boolean isString() const;
```

If the data in the column is a pointer to a character string or an IString, true is returned.

isTime

```
Boolean isTime() const;
```

If the data in the column is a time, true is returned.

isVisible

```
Boolean isVisible() const;
```

If the column is visible in the container, true is returned.

isWriteable

Condition: (1) Ignored on AIX

```
Boolean isWriteable() const;
```

If the user can edit the data in the column, true is returned.

justifyData

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &justifyData(
                            VerticalAlignment = centeredVertically,
                            HorizontalAlignment = centered);
```

Sets the justification of the data to a horizontal and vertical alignment. Use the enumeration HorizontalAlignment (see page 146) to set the horizontal alignment to left,

right, or centered. VerticalAlignment (see page 146) sets the vertical alignment to top, bottom, or centeredVertically.

justifyHeading

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &justifyHeading(
                            VerticalAlignment = centeredVertically,
                            HorizontalAlignment = centered);
```

Sets the justification of the heading to a horizontal and vertical alignment. Use the enumeration HorizontalAlignment (see page 146) to set the horizontal alignment to left, right, or centered. VerticalAlignment (see page 146) sets the vertical alignment to top, bottom, or centeredVertically.

setDataOffset

```
virtual IContainerColumn &setDataOffset(
                            unsigned long dataOffset);
```

Identifies where the data is located in a container object for this column.

The data in the object and the data displayed in the column are connected. For a user to have a details view, you must derive a class from IContainerObject (see page 209) and extend it with the additional “fields” to display in the columns of the container. You must ensure the column object has the following information:

- The exact offset of the data in the object
- The type of the data in the object

The column object must have the correct information because it handles the drawing of this data. If the column object does not have the correct information, the behavior of the column is undefined.

setDefaultDataStyle

```
static void setDefaultDataStyle(const DataStyle &dataStyle);
```

Sets the default data style for all subsequent column data.

setDefaultHeadingStyle

```
static void setDefaultHeadingStyle(
             const HeadingStyle &headingStyle);
```

Sets the default heading style for all subsequent column headings.

setDisplayWidth

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setDisplayWidth(
                            unsigned long widthInPixels);
```

Returns the displayable width of the column, in pixels, after showing the details view.

If you want the column object's width, use IContainerControl::detailObjectRectangle instead of IContainerColumn::displayWidth. This is because a container details view cell is like a rectangle inside of another rectangle:

- The inner rectangle is the data rectangle of the cell and does not contain the required column margin.

  This rectangle's width is either:
  - The value set via IContainerColumn::setDisplayWidth
  - The value queried via IContainerColumn::displayWidth
- The outer rectangle is the complete cell, including the margins that the container control requires.

This is the value returned by IContainerControl::detailObjectRectangle.

setHeadingIcon

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setHeadingIcon(
                            const IPointerHandle &iconHandle);
```

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setHeadingIcon(
                            const IResourceId &iconId);
```

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setHeadingIcon(
                            unsigned long iconId);
```

Sets the icon to use for the heading and changes the style to icon.

setHeadingText

```
virtual IContainerColumn &setHeadingText(
                            const IResourceId &textId);

virtual IContainerColumn &setHeadingText(const char *text);
```

Sets the text to use for the heading and changes the style to string.

setHelpId

```
virtual IContainerColumn &setHelpId(unsigned long helpId);
```

Stores a help panel ID. You set the help panel ID by calling the function IContainerColumn::setHelpId. If you have attached an ICnrHandler to the container, the handler displays this help panel when the user requests help while direct-editing the column data of the details view.

show

```
virtual IContainerColumn &show(Boolean visible = true);
```

Makes the column visible.

showSeparators

Condition: (1) Ignored on AIX

```
virtual IContainerColumn &showSeparators(
                            const DataStyle &separatorStyles = horizontalSeparator ] verticalSeparator);
```

Adds either:

- The vertical separator after a column
- The horizontal separator under the column heading
- Both of the preceding separators

By default, this function adds both.

verticalDataAlignment

Condition: (1) Ignored on AIX

```
VerticalAlignment verticalDataAlignment() const;
```

Returns the current alignment of column data in a container details view. The returned value is an enumerator provided by VerticalAlignment (see page 146).

verticalHeadingAlignment

Condition: (1) Ignored on AIX

```
VerticalAlignment verticalHeadingAlignment() const;
```

Returns the current alignment of column heading data in a container details view. The returned value is an enumerator provided by VerticalAlignment (see page 146).

Figure 51. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

columnInfo
```
_FIELDINFO *columnInfo() const;
```

Returns the address of the container's column record.

container
```
IContainerControl *container() const;
```

Returns the container in which this column is located.

dataAttributes
Condition: (1) Ignored on AIX

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

Returns the current data attributes in the container.

invalidate
```
IContainerColumn &invalidate();
```

If the column is in a container, the column is refreshed.

setColumnInfo
```
IContainerColumn &setColumnInfo(_FIELDINFO *fieldinfo);
```

Stores the address of the container's column record.

setContainer
```
IContainerColumn &setContainer(
                    IContainerControl *container);
```

Stores the specified container, which is the container this column is located in.

setDataAttributes
Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setDataAttributes(
                            unsigned long dataAttributes);
```

Sets the specified attributes into the data.

setTitleAttributes
Condition: (1) Ignored on AIX

```
virtual IContainerColumn &setTitleAttributes(
                            unsigned long titleAttributes);
```

Sets the specified attributes into the title.

titleAttributes

Condition: (1) Ignored on AIX

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

Returns the current title attributes in the container.

Enumerations

DataSource

```
public:
enum DataSource
  {
  isIcon,
  isIconViewText,
  };
```

These enumerators specify whether the column data is the icon or the icon text stored in the object:

isIcon
Uses the icon from the icon view.

isIconViewText
Uses the text from the icon view.

HorizontalAlignment

```
public:
enum HorizontalAlignment
  {
  left,
  right,
  centered,
  };
```

These enumerators specify the horizontal alignment of text in the column:

left
Left-justifies the text in the column.

right
Right-justifies the text in the column.

centered
Centers the text horizontally in the column.

VerticalAlignment

```
public:
enum VerticalAlignment
  {
  top,
  bottom,
  centeredVertically,
  };
```

These enumerators specify the vertical alignment of text in the column:

top
Aligns the text at the top of the column.

bottom
Aligns the text at the bottom of the column.

centeredVertically
Centers the text vertically in the column.

Styles

DataStyle

These flags represent construction values for the data in a column. In addition to the IContainerColumn::Style types, the data in a column can also contain the following data styles:

classDefaultDataStyle
Specifies the original default style for data in a column, which is string, alignVerticallyCentered, alignLeft, and readOnly.

date
Displays the data in date format with National Language Support enabled. The date format is a four-byte field divided in the following order:

- A single byte for the day
- A single byte for the month
- Two bytes for the year

Note: In OS/2, this format is the same as the CDATE structure. For additional information, see *Presentation Manager Programming Reference Volume 3*.

handleDrawItem
When a details item must be drawn, causes an ICnrDrawItemEvent to be dispatched to ICnrDrawHandler::drawDetailsItem.

horizontalSeparator
Draws a horizontal separator beneath column headings.

invisible
Hides the data in the column by making it invisible.

number
Specifies the data is an unsigned long number.

readonly
Disables editing of the data in the column.

time
Displays the data in a time format with National Language Support enabled. The time format is a four-byte field divided in the following order:

- A single byte for the hours

- A single byte for the minutes
- A single byte for the seconds
- A final byte, which is reserved

Note: In OS/2, this format is the same as the CTIME structure. For additional information, see *Presentation Manager Programming Reference Volume 3*.

verticalSeparator
Draws a vertical separator after the column.

HeadingStyle

These flags represent construction values for the column heading. In addition to the IContainerColumn::Style types, the heading can also contain the following styles:

classDefaultHeadingStyle
Specifies the original default style for a heading, which is string, alignVerticallyCentered, alignLeft, and readOnlyHeading.

readOnlyHeading
Disables editing the data in the heading.

Style

These flags represent construction values for both the column heading and the column data:

alignBottom
Aligns the data to the bottom of the cell.

alignCentered
Centers the data horizontally in the cell.

alignLeft
Aligns the data to the left of the cell.

alignRight
Aligns the data to the right of the cell.

alignTop
Aligns the data to the top of the cell.

alignVerticallyCentered
Centers the data vertically in the cell.

icon
Specifies the data is an icon.

string
Specifies the data is a string.

Chapter 37. IContainerControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IContainerControl

Header File: icnrctl.hpp

Objects of the IContainerControl class display a container of objects in any of the following supported views:

- Icon
- Name
- Tree
- Details
- Text

Like other controls in the library, you can construct this control in one of the following ways:

- On a window
- Loaded in a dialog template
- As a pre-existing control

Motif Information: In AIX, due to the limited capabilities of the container widget under Motif 1.2, the User Interface Class Library does not support the following:

- Drag and drop support via the IDM* classes.
- Direct editing.
- Vertical split bar in a details view.
- Container titles. The library does support details view column titles.
- Setting refresh on and off.
- Any emphasis except selection emphasis
- Draw item, for example, owner draw.
- Window and workspace coordinates. There is no support for placing IContainerObjects at explicit x,y locations.
- Scrolling functions, for example:
  - scroll
  - scrollDetailsHorizontally
  - scrollHorizontally
  - scrollVertically
  - scrollToObject
- The IContainerControl constuctors that create an object from an existing container control or a dialog template.
- The ICnrHandler overridden member functions that the library does not call:
  - cursoredChanged

© Copyright IBM Corp. 1994

– deltaReached
– inuseChanged
– windowScrolled

Nested Classes

IContainerControl::Style
IContainerControl::Attribute
IContainerControl::Iterator
IContainerControl::ObjectCursor
IContainerControl::FilterFn
IContainerControl::TextCursor
IContainerControl::CompareFn
IContainerControl::ColumnCursor

Constructors

```
public:
IContainerControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &location = IRectangle ( ),
   const Style &style = defaultStyle ( ),
   const Attribute &attribute = defaultAttribute ( ));
```

Condition: (2) OS/2 Only

```
public:
IContainerControl(unsigned long id, IWindow *parentDialog);
```

Condition: (2) OS/2 Only

```
public:
IContainerControl(const IWindowHandle &handle);

protected:
IContainerControl();
```

You can construct objects of this class in the following ways:

- On a window by specifying the parent and owner
- From a container control on a loaded dialog template
- From an existing container control's window handle

Note: The AIX version does not support constructing objects from a loaded dialog template or from an existing container control's window handle.

Public Members

addColumn

```
virtual IContainerControl &addColumn(
                              const IContainerColumn *column,
                              const IContainerColumn *afterColumn = 0);
```

Adds a column to the container. If you specify *afterColumn*, the container adds the column after this column. Otherwise, the container adds the column as the last column.

addObject

```
virtual IContainerControl &addObject(
                              const IContainerObject *newObject,
                              IContainerObject *parentObject = 0);
```

Adds an object to the container. In the tree view, the parent defines the object's location. When you specify *parentObject*, this function adds the object as its last immediate descendent. Otherwise, the container adds the object as the last object in the root level.

addObjectAfter

```
virtual IContainerControl &addObjectAfter(
                              const IContainerObject *newObject,
                              const IContainerObject *afterObject,
                              IContainerObject *parentObject = 0);
```

Adds a new object by placing it after a specified object.

You can add a container object as the first object in the container by specifying 0 for *afterObject*. This places the object at either of the following locations:

- The beginning of the container
- If you specify *parentObject*, the beginning of the subtree specified by this parameter
- After a specified object in the same level of the tree

If *afterObject* is nonzero, the container ignores *parentObject*. The level, in this case, is determined by afterObject.

allObjectsDo

```
virtual IContainerControl &allObjectsDo(
                              IContainerControl::Iterator &iteratorObject,
                              Boolean includeDescendents = false);
```

Calls IContainerControl::Iterator::applyTo (see page 200) on the specified iterator for all objects in the container. If you specify true for *includeDescendents*, the tree view descendent objects are called as well. The iteration stops when:

- IContainerControl::Iterator::applyTo returns true
- All the objects in the container have been processed

areDetailsViewTitlesVisible

Condition: (1) Ignored on AIX

```
Boolean areDetailsViewTitlesVisible() const;
```

Queries whether the column titles are currently displayed for details view.

arrangeIconView

```
virtual IContainerControl &arrangeIconView();
```

Arranges the icon view. The icons are arranged in horizontal rows from left to right.

Presentation Manager Information: CM_ARRANGE provides additional information on arrangement in the *Presentation Manager Programming Reference Volume 3*.

closeEdit

Condition: (1) Ignored on AIX

```
virtual IContainerControl &closeEdit();
```

Closes an open edit field.

collapse

Condition: (1) Ignored on AIX

```
virtual IContainerControl &collapse(
                            IContainerObject *object);
```

Collapses the descendents of an object in the tree view.

Note: The parent is not actually being collapsed or expanded.

collapseTree

Condition: (1) Ignored on AIX

```
virtual IContainerControl &collapseTree();
```

Collapses all nodes of a tree view.

columnAt

```
IContainerColumn *columnAt(unsigned long index) const;

IContainerColumn *columnAt(
                     const ColumnCursor &cursor) const;
```

Retrieves the column at the specified 0-based index or cursor.

columnCount

```
unsigned long columnCount() const;
```

Returns the number of columns in the container.

columnUnderPoint

Condition: (1) Ignored on AIX

```
IContainerColumn *columnUnderPoint(
                    const IPoint &point) const;
```

Retrieves the column under the specified point, in window coordinates.

containerFromHandle

```
static IContainerControl *containerFromHandle(
                            const IWindowHandle &handle);
```

Retrieves a container from the list using the specified handle.

containsObject

```
Boolean containsObject(
          const IContainerObject *object) const;
```

If the specified object is in the container, true is returned.

convertToWorkspace

Condition: (1) Ignored on AIX

```
IRectangle convertToWorkspace(
             const IRectangle &windowRectangle,
             Boolean rightWindow = false) const;
```

Converts a rectangle from container window coordinates to workspace coordinates. This is valid only for the current view.

copyObjectTo

```
virtual IContainerObject *copyObjectTo(
                            IContainerObject *copyObject,
                            IContainerObject *parentObject = 0,
                            IContainerControl *newContainer = 0,
                            IContainerObject *afterObject = 0,
                            const IPoint &iconViewLocation = IPoint ( 0 , 0 ));
```

Copies an object and its descendents to a new location in the tree view. The object must override IContainerObject::objectCopy (see page 214). This function returns a copy of the object.

Presentation Manager Information: This function is called during drag and drop. It must be overridden for an object to support drag and drop. If you specify 0 for *afterObject*, the copied object is added as the first object in the tree view.

currentEditColumn

Condition: (1) Ignored on AIX

```
IContainerColumn *currentEditColumn() const;
```

Retrieves the column being edited. If a column is not being edited, 0 is returned.

currentEditMLE

Condition: (1) Ignored on AIX

```
IMultiLineEdit *currentEditMLE() const;
```

Retrieves the multiple-line edit (MLE) field being used for editing. If an edit operation is not in effect, 0 is returned.

currentEditObject

Condition: (1) Ignored on AIX

```
IContainerObject *currentEditObject() const;
```

Retrieves the object being edited. If an object is not being edited, 0 is returned.

cursoredObject

Condition: (1) Ignored on AIX

```
virtual IContainerObject *cursoredObject() const;
```

Returns the object on which the cursor is located.

defaultAttribute

```
static Attribute defaultAttribute();
```

Returns the default attribute. The default attribute is classDefaultAttribute (see page 191) unless you have changed it using setDefaultAttribute (see page 175).

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 204) unless you have changed it using setDefaultStyle (see page 175).

deleteAllObjects

```
virtual IContainerControl &deleteAllObjects();
```

Deletes all objects in this container. If an object exists in other containers, it is removed from them as well before being deleted.

deleteSelectedObjects

```
virtual IContainerControl &deleteSelectedObjects();
```

Deletes the selected objects in this container. If an object exists in other containers, it is removed from them as well before being deleted.

descendentsOf

```
ICnrObjectSet descendentsOf(
               IContainerObject *parentObject) const;
```

Returns the set of all descendents of an object in the tree view.

detailObjectRectangle

Condition: (1) Ignored on AIX

```
IRectangle detailObjectRectangle(
            const IContainerObject *object,
            Boolean rightWindow = false) const;
```

Condition: (1) Ignored on AIX

```
IRectangle detailObjectRectangle(
            const IContainerObject *object,
            const IContainerColumn *column) const;
```

Returns the details view rectangle. If you specify a column, this function provides the cell in the details view. This rectangle is in container window coordinates.

If you want the column object's width, use IContainerControl::detailObjectRectangle instead of IContainerColumn::displayWidth. This is because a container details view cell is like a rectangle inside of another rectangle where:

- The inner rectangle is the data rectangle of the cell and does not contain the required column margin.

  This rectangle's width is either:

  – The value set via IContainerColumn::setDisplayWidth
  – The value queried via IContainerColumn::displayWidth

  This is the value returned by IContainerControl::detailObjectRectangle.
- The outer rectangle is the complete cell, including the margins that the container control requires.

The parameters are:

object The object you want the details view rectangle for.

rightWindow
If the container is in the details view, use this flag to specify which rectangle to return. If you specify true, the rectangle is for the right window. Otherwise, the rectangle is for the left window.

column A column identifying a boundary of a rectangle.

Note: This function is obsolete. Use IContainerControl::detailsObjectRectangle instead.

detailsObjectRectangle

Condition: (1) Ignored on AIX

```
IRectangle detailsObjectRectangle(
            const IContainerObject *object,
            const IContainerColumn *column) const;
```

Condition: (1) Ignored on AIX

```
IRectangle detailsObjectRectangle(
            const IContainerObject *object,
            Boolean rightWindow = false) const;
```

Returns the details view rectangle. If you specify a column, this function provides the cell in the details view. This rectangle is in container window coordinates.

If you want the column object's width, use IContainerControl::detailsObjectRectangle instead of IContainerColumn::displayWidth. This is because a container details view cell is like a rectangle inside of another rectangle:

- The inner rectangle is the data rectangle of the cell and does not contain the required column margin.

  This rectangle's width is either:
  - The value set via IContainerColumn::setDisplayWidth
  - The value queried via IContainerColumn::displayWidth
- The outer rectangle is the complete cell, including the margins that the container control requires.

This is the value returned by IContainerControl::detailsObjectRectangle.

The parameters are:

object The object you want the details view rectangle for.

rightWindow
If the container is in the details view, use this flag to specify which rectangle to return. This will be the rectangle that encompasses all columns for an object in either the left or right details view window. If you specify true, the rectangle is for the right window. Otherwise, the rectangle is for the left window.

column A column identifying a boundary of a rectangle.

detailsTitleRectangle

Condition: (1) Ignored on AIX

```
IRectangle detailsTitleRectangle(
            Boolean rightSide = false) const;
```

Returns the details view, column-title rectangle in container window coordinates. This rectangle represents all currently visible column titles in the details view window, not just one column. The container must be in details view when this function is called.

detailsViewPortOnWindow

Condition: (1) Ignored on AIX

```
IRectangle detailsViewPortOnWindow(
            Boolean rightSide = false) const;
```

Retrieves either the right or left work area of a split details view, in window coordinates.

detailsViewPortOnWorkspace

Condition: (1) Ignored on AIX

```
IRectangle detailsViewPortOnWorkspace(
            Boolean rightSide = false) const;
```

Retrieves either the right or left work area of a split details view, in workspace coordinates.

detailsViewSplit

Condition: (1) Ignored on AIX

```
IContainerColumn *detailsViewSplit() const;
```

Returns the last column before the split bar.

disableCaching

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableCaching();
```

Stops the dispatch of ICnrQueryDeltaEvents (see page 97) to an ICnrHandler (see page 87).

disableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableDataUpdate(
                              IContainerObject *object);
```

Disables direct editing of the text of an object.

disableDrawBackground

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableDrawBackground();
```

Stops the dispatch of ICnrDrawBackgroundEvent to an ICnrDrawHandler.

disableDrawItem

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableDrawItem();
```

Stops the dispatch of ICnrDrawBackgroundEvent to an ICnrDrawHandler.

disableDrop

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableDrop(
                            IContainerObject *object);
```

Disables an object from receiving a drop. No target emphasis is drawn.

disableTitleUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerControl &disableTitleUpdate();
```

Disables the container title from being edited.

editColumnTitle

Condition: (1) Ignored on AIX

```
virtual IContainerControl &editColumnTitle(
                            IContainerColumn *column);
```

Opens an edit field on a specified column's heading.

editContainerTitle

Condition: (1) Ignored on AIX

```
virtual IContainerControl &editContainerTitle();
```

Opens an edit field on the container title.

editObject

Condition: (1) Ignored on AIX

```
virtual IContainerControl &editObject(
                            IContainerObject *object,
                            IContainerColumn *column = 0);
```

Opens an edit field on a specified object.

enableCaching

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableCaching(
                            unsigned long ulDeltaValue = 30);
```

If an ICnrHandler (see page 87) has been created and added to the container, this function causes the dispatch of an ICnrQueryDeltaEvent (see page 97) to the handler when the container scrolls past the pre-defined delta value.

enableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableDataUpdate(
                            IContainerObject *object,
                            Boolean enable = true);
```

Enables direct editing of the text of a specified object. In addition, the container itself must be in a state that allows direct editing. You can do this by constructing the container without the style IContainerControl::readOnly.

enableDrawBackground

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableDrawBackground(
                            Boolean Enable = true);
```

If an ICnrDrawHandler has been created and added to the container, this function causes the container to dispatch the ICnrDrawBackgroundEvent to its handlers. Typically, you process this event by deriving a handler from ICnrDrawHandler and overriding the function drawBackground.

enableDrawItem

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableDrawItem(
                            Boolean Enable = true);
```

If an ICnrDrawHandler has been created and added to the container, this function causes the container to dispatch the ICnrDrawBackItemEvent to its handlers. Typically, you process this event by deriving a handler from ICnrDrawHandler and overriding the functions of the handler such as drawText and drawIcon.

enableDrop

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableDrop(
                            IContainerObject *object,
                            Boolean enable = true);
```

Enables a specified object to be the target of a drag event.

enableNotification

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

Causes the container to send notifications to its observer objects.

enableTitleUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerControl &enableTitleUpdate(
                            Boolean enable = true);
```

Allows the container title to be edited.

expand

Condition: (1) Ignored on AIX

```
virtual IContainerControl &expand(IContainerObject *object);
```

Expands the tree view to show the descendents of the specified parent object.

expandTree

Condition: (1) Ignored on AIX

```
virtual IContainerControl &expandTree();
```

Expands all nodes of a tree view.

filter

```
virtual IContainerControl &filter(
                            const IContainerControl::FilterFn &filterObject);

virtual IContainerControl &filter();
```

Filters the container by calling the specified IContainerControl::FilterFn::isMemberOf (see page 199) function for each object in the container. If you do not specify a FilterFn, objects currently hidden or filtered from the container are restored to the container.

hasMixedTargetEmphasis

Condition: (1) Ignored on AIX

```
Boolean hasMixedTargetEmphasis() const;
```

Queries if the mixed target emphasis mode is set for drag operations over this container.

hasNormalTargetEmphasis

Condition: (1) Ignored on AIX

```
Boolean hasNormalTargetEmphasis() const;
```

Queries if the regular target emphasis mode is set for drag operations over this container.

hasOrderedTargetEmphasis

Condition: (1) Ignored on AIX

```
Boolean hasOrderedTargetEmphasis() const;
```

Queries if the ordered target emphasis mode is set for drag operations over this container.

hideDetailsViewTitles

```
virtual IContainerControl &hideDetailsViewTitles();
```

Removes column headings from details view.

hideObject

```
virtual IContainerControl &hideObject(
                            IContainerObject *object);
```

Makes a specified object invisible by filtering it out of the container.

hideSourceEmphasis

Condition: (1) Ignored on AIX

```
virtual IContainerControl &hideSourceEmphasis();
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &hideSourceEmphasis(
                            IContainerObject *object);
```

Removes source emphasis. If you specify an object, this function removes the emphasis from that object. Otherwise, it removes source emphasis from the container itself.

hideSplitBar

Condition: (1) Ignored on AIX

```
virtual IContainerControl &hideSplitBar();
```

Removes the split bar from the details view work area.

hideTitle

Condition: (1) Ignored on AIX

```
virtual IContainerControl &hideTitle();
```

Removes the container title from the container.

hideTitleSeparator

Condition: (1) Ignored on AIX

```
virtual IContainerControl &hideTitleSeparator();
```

Removes the title separator from the container window.

hideTreeLine

```
virtual IContainerControl &hideTreeLine();
```

Does not show the lines connecting parents to children's records.

iconRectangle

Condition: (1) Ignored on AIX

```
IRectangle iconRectangle(
            const IContainerObject *object,
            Boolean includeText = false) const;
```

Returns the rectangle bounding an object's icon. If you specify *includeText* as true, this rectangle also includes the object's text.

iconSize

Condition: (1) Ignored on AIX

```
ISize iconSize() const;
```

Returns the icon or bit-map size of all objects.

immediateDescendentsOf

```
ICnrObjectSet immediateDescendentsOf(
               IContainerObject *parentObject) const;
```

Returns the set of immediate descendents *only* of the specified object in the tree view.

initialize

```
static void initialize();
```

Sets up the container environment. If the first container component you create is an IContainerControl, the library calls this function automatically.

isCachingEnabled

Condition: (1) Ignored on AIX

```
Boolean isCachingEnabled() const;
```

Queries whether a delta value is set and caching is enabled.

isCollapsed

Condition: (1) Ignored on AIX

```
Boolean isCollapsed(const IContainerObject *object) const;
```

Queries whether the specified tree-view node object is currently collapsed.

isColumnRight

Condition: (1) Ignored on AIX

```
Boolean isColumnRight(const IContainerColumn *column) const;
```

If the specified column is to the right of the split bar, true is returned.

isCursored

Condition: (1) Ignored on AIX

```
Boolean isCursored(const IContainerObject *object) const;
```

Queries whether the specified object has cursored emphasis.

isDetailsView

```
Boolean isDetailsView() const;
```

Queries whether the container is currently in the details view.

isDrawBackgroundEnabled

Condition: (1) Ignored on AIX

```
Boolean isDrawBackgroundEnabled() const;
```

Queries whether owner draw of the container background is enabled.

isDrawItemEnabled

Condition: (1) Ignored on AIX

```
Boolean isDrawItemEnabled() const;
```

Queries whether owner draw of container objects is enabled.

isDropOnAble

Condition: (1) Ignored on AIX

```
Boolean isDropOnAble(const IContainerObject *object) const;
```

Queries whether the specified object is set to receive a direct manipulation drop.

isExpanded

Condition: (1) Ignored on AIX

```
Boolean isExpanded(const IContainerObject *object) const;
```

Queries whether the specified tree-view node object is currently expanded.

isExtendedSelection

```
Boolean isExtendedSelection() const;
```

Queries whether the container is in extended selection mode. Extended selection is an enhanced version of single selection that lets the user select discontiguous sets of container items.

isFlowed

```
Boolean isFlowed() const;
```

Queries whether the current container view is flowed. This function applies to the name and text views only.

isFlowedNameView

```
Boolean isFlowedNameView() const;
```

Returns true if the container is currently in flowed name view.

isFlowedTextView

```
Boolean isFlowedTextView() const;
```

Returns true if the container is currently in flowed text view.

isIconView

```
Boolean isIconView(Boolean iconOnly = false) const;
```

Queries whether the container is currently in an icon view. If *iconOnly* is false, true is returned for both icon view and tree-icon view. If *iconOnly* is true, true is returned only for icon view.

isInUse

Condition: (1) Ignored on AIX

```
Boolean isInUse(const IContainerObject *object) const;
```

Queries whether the specified object has in-use emphasis.

isMoveValid

```
virtual Boolean isMoveValid(
                  IContainerObject *moveObject,
                  IContainerObject *newParentObject = 0,
                  IContainerControl *newContainer = 0,
                  IContainerObject *afterObject = 0);
```

Determines whether an object and its descendents can be moved to a new location. This function checks the validity of a move to be performed by IContainerControl::moveObjectTo (see page 168).

The parameters are the following:

moveObject
: The object and all of its descendents (if any) you want to move.

newParentObject
: The new parent for *moveObject*. The default is *moveObject* becomes a root-level object (that is, there is no parent).

newContainer
: The container that is the target of the move operation. The default is the container you are calling this function upon (that is, the source container).

afterObject
: The object in the target container after which you want to place *moveObject*. If *newParentObject* is 0 and *afterObject* is 0, *moveObject* is the first object in the container. If *newParentObject* is not 0 and *afterObject* is 0, *moveObject* is the first child of *newParentObject*.

In the following list, the source container is the container you are calling this function upon. If you specify 0 for the target container, it is the same container as the souce. Otherwise, it is another container.

This function returns false for any of the following reasons:

- *moveObject* = 0.
- The source container does not contain *moveObject*.
- The target container does not contain a valid *afterObject* (that is, nonzero).
- The target container does not contain a valid *newParentObject* (that is, nonzero).
- You specify both afterObject and *newParentObject* (that is, they are both nonzero), and *newParentObject* is not the parent of *afterObject* in the target container.
- The source and target containers are the same and you specified *moveObject* to become a child of one of its descendents. You cannot move a subtree to become a child of one of the subtree's members.
- The source and target containers are different and *moveObject* or any of its descendents already exist in the target container. This is because you can share container objects between containers.

isMultipleSelection

```
Boolean isMultipleSelection() const;
```

Queries whether the container is in multiple selection mode. Multiple selection is a mode that allows the user unrestricted selection of objects.

isNameView

```
Boolean isNameView(Boolean nameOnly = false) const;
```

Queries whether the container is currently in a name view. If *nameOnly* is false, true is returned for both name view and tree-name view. If *nameOnly* is true, true is returned for only name view.

isReadOnly

Condition: (1) Ignored on AIX

```
Boolean isReadOnly(const IContainerObject *object) const;
```

Queries whether the specified object is read-only.

Note: This function is obsolete. Use IContainerControl::isWriteable instead.

isRefreshOn

Condition: (1) Ignored on AIX

```
Boolean isRefreshOn() const;
```

Queries the refresh state of the container. If the container immediately displays visual changes, true is returned.

isSelected

```
Boolean isSelected(const IContainerObject *object) const;
```

Queries whether the specified object has selection emphasis.

isSingleSelection

```
Boolean isSingleSelection() const;
```

Queries whether the container is in the single selection mode. In this mode, the user (or your code) can only select a single object at a time. Selecting a new object removes the selection from a prior object. In a single-selection container one object is always selected. If you remove an object with selection emphasis, the container selects another object.

isSource

Condition: (1) Ignored on AIX

```
Boolean isSource(const IContainerObject *object) const;
```

Queries whether the specified object has source emphasis. Source emphasis is a visual cue that the object is the source of an operation. Objects being dragged and objects with pop-up menus normally display source emphasis.

isTarget

Condition: (1) Ignored on AIX

```
Boolean isTarget(const IContainerObject *object) const;
```

Queries whether the specified object is a target of direct manipulation.

isTextView

```
Boolean isTextView(Boolean textOnly = false) const;
```

Queries whether the container is currently in a text view. If *textOnly* is false, true is returned for both text view and tree-text view. If *textOnly* is true, true is returned for only text view.

isTitleSeparatorVisible

Condition: (1) Ignored on AIX

```
Boolean isTitleSeparatorVisible() const;
```

Queries whether the title separator is currently displayed.

isTitleVisible

Condition: (1) Ignored on AIX

```
Boolean isTitleVisible() const;
```

Queries whether the container title is currently displayed.

isTitleWriteable

Condition: (1) Ignored on AIX

```
Boolean isTitleWriteable() const;
```

Returns true if the data in the title can be edited.

isTreeIconView

```
Boolean isTreeIconView() const;
```

Returns true only if the container is in tree-icon view.

isTreeNameView

```
Boolean isTreeNameView() const;
```

Returns true only if the container is currently in tree-name view.

isTreeTextView

```
Boolean isTreeTextView() const;
```

Returns true only if the container is currently in tree-text view.

isTreeView

```
Boolean isTreeView() const;
```

Queries whether the container is currently in a tree view. This includes the tree-icon, the tree-text, and the tree-name views.

isVisible

```
virtual Boolean isVisible() const;
```

Condition: (1) Ignored on AIX

```
Boolean isVisible(const IContainerObject *object) const;
```

The isVisible function that does not use an object for a parameter is an override of IWindow::visible, which queries whether the container is visible.

If an object is specified, the function queries whether the object is visible.

isWriteable

Condition: (1) Ignored on AIX

```
Boolean isWriteable(const IContainerObject *object) const;
```

Returns true if the object can be edited.

lineSpacing

Condition: (1) Ignored on AIX

```
long lineSpacing() const;
```

Returns the space between lines, in pixels. This is the amount you can add to the minimum distance required by the container control for emphasis painting. You cannot reduce the minimum distance required by the presentation system.

moveIconTo

Condition: (1) Ignored on AIX

```
virtual IContainerControl &moveIconTo(
                            IContainerObject *object,
                            const IPoint &point);
```

Moves the specified object to a new workspace location within the icon view.

moveObjectTo

```
virtual Boolean moveObjectTo(
                  IContainerObject *moveObject,
                  IContainerObject *newParentObject = 0,
                  IContainerControl *newContainer = 0,
                  IContainerObject *afterObject = 0,
                  const IPoint &iconViewLocation = IPoint ( 0 , 0 ));
```

Moves the specified object and its descendents to the specified new location. This function calls IContainerControl::isMoveValid (see page 165) to determine if the move is possible. If an object and its descendents do not already exist in the new location, this function moves the object and its descendents to the new container. Returns true if the move was valid and successful.

The parameters are the following:

moveObject
: The object and all of its descendents (if any) you want to move.

newParentObject
: The new parent for *moveObject*. The default is *moveObject* becomes a root-level object (that is, there is no parent).

newContainer
: The container that is the target of the move operation. The default is the container you are calling this function upon (that is, the source container).

afterObject
: The object in the target container after which you want to place *moveObject*. If *newParentObject* is 0 and *afterObject* is 0, *moveObject* is the first object in the container. If *newParentObject* is not 0 and *afterObject* is 0, *moveObject* is the first child of *newParentObject*.

iconViewLocations
: The icon view position in workspace coordinates.

nlsCompare

```
static long nlsCompare(
             const char *text1,
             const char *text2);
```

Compares two specified strings consisting of a national character set. Use this function for implementing National Language Support (NLS).

numberOfColumnChanges

```
unsigned long numberOfColumnChanges() const;
```

Retrieves the container's count of column additions and removals. IContainerControl::ColumnCursor objects use this function to validate their location in the container.

numberOfObjectChanges

```
unsigned long numberOfObjectChanges() const;
```

Retrieves the container's count of object additions and removals. IContainerControl::ObjectCursor and IContainerControl::TextCursor objects use this function to validate their location in the container.

objectAt

```
virtual IContainerObject *objectAt(
                            unsigned long index) const;

virtual IContainerObject *objectAt(
                            const IContainerControl::ObjectCursor &cursor) const;
```

Condition: (1) Ignored on AIX

```
virtual IContainerObject *objectAt(
                            const IContainerControl::TextCursor &cursor) const;
```

Returns an object at the specified index or cursor position in the container.

objectCount

```
unsigned long objectCount() const;
```

Returns the number of objects in the container.

objectList

```
ICnrObjectSet objectList() const;
```

Returns the set of all objects in the container.

objectUnderPoint

Condition: (1) Ignored on AIX

```
IContainerObject *objectUnderPoint(
                    const IPoint &point) const;
```

Retrieves the object under the specified point, in window coordinates. If the container is in the icon view, this function returns the top-most object.

operator ==

```
Boolean operator ==(const IContainerControl &that);
```

If two containers are the same (that is, their storage location is identical), true is returned.

parentObject

```
virtual IContainerObject *parentObject(
                            const IContainerObject *childObject) const;
```

Returns the parent of an object in the tree view. If the object does not have a parent object, true is returned.

refresh

Condition: (1) Ignored on AIX

```
virtual IContainerControl &refresh(Boolean refreshChildren);
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &refresh();
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &refresh(
                            IContainerObject *object);
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &refresh(
                            const IRectangle &invalidRectangle);
```

Refreshes all objects in the container.

refreshAllContainers

Condition: (1) Ignored on AIX

```
static void refreshAllContainers();
```

Refreshes all containers.

removeAllObjects

```
virtual IContainerControl &removeAllObjects();
```

Removes all objects from the container.

removeColumn

```
virtual IContainerControl &removeColumn(
                            const IContainerColumn *column);
```

Removes the specified column from the container.

removeColumnAt

```
virtual IContainerControl &removeColumnAt(
                            IContainerControl::ColumnCursor &cursor);
```

Removes the column at a specified cursor position.

removeInUse

Condition: (1) Ignored on AIX

```
virtual IContainerControl &removeInUse(
                            IContainerObject *object);
```

Removes in-use emphasis from the specified object.

removeObject

```
virtual IContainerControl &removeObject(
                            IContainerObject *object,
                            Boolean fAllContainers = false);
```

Removes the specified object from this container or, optionally, all containers.

removeObjectAt

```
virtual IContainerControl &removeObjectAt(
                            IContainerControl::ObjectCursor &cursor);

virtual IContainerControl &removeObjectAt(
                            IContainerControl::TextCursor &cursor);
```

Removes the object at the specified cursor's position.

removeSelected

```
virtual IContainerControl &removeSelected(
                            IContainerObject *object);
```

Removes selection emphasis from the specified object.

removeSelectedObjects

```
virtual IContainerControl &removeSelectedObjects();
```

Removes all selected objects from the container.

scroll

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scroll(
                            long vericalPixels,
                            long horizontalPixels,
                            Boolean rightSide = false);
```

Scrolls the container both horizontally and vertically. If you specify a nonzero *horizontalPixels* and the container is currently displaying a split details view, *rightSide* determines which pane is scrolled horizontally.

The parameters are:

verticalPixels
: The number of pixels to scroll in the vertical direction.

horizontalPixels
: The number of pixels to scroll in the horizontal direction.

rightSide
: If this container is in details view, specify true to scroll the right pane and false to scroll the left pane.

scrollDetailsHorizontally

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scrollDetailsHorizontally(
                            long horizontalPixels,
                            Boolean rightSide = false);
```

Scrolls either the right or left side of a split details view horizontally.

The parameters are:

horizontalPixels
The number of pixels to scroll in the horizontal direction.

rightSide Specifies which side to scroll. True indicates the right pane and false indicates the left pane.

scrollHorizontally

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scrollHorizontally(
                            long pixels,
                            Boolean rightSide = false);
```

Scrolls the container horizontally.

The parameters are:

pixels The number of pixels to scroll horizontally.

rightSide If this container is in details view, specify true to scroll the right pane and false to scroll the left pane.

scrollToObject

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scrollToObject(
                            const IContainerObject *object);
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scrollToObject(
                            const IContainerObject *object,
                            const IContainerColumn *column,
                            Boolean leftJustify = true);
```

Scrolls the container to the specified object and, optionally, a specified column into the work area. If you specify false for *leftJustify*, the column is scrolled into the work area at the closest border.

scrollVertically

Condition: (1) Ignored on AIX

```
virtual IContainerControl &scrollVertically(long pixels);
```

Scrolls the container vertically.

sendEvent

Condition: (3) AIX only

```
virtual IEventResult sendEvent(
                      EventType eventType,
                      const IEventParameter1 &parm1 = 0,
                      const IEventParameter2 &parm2 = 0) const;
```

Condition: (3) AIX only

```
virtual IEventResult sendEvent(
                      unsigned long eventId,
                      const IEventParameter1 &parm1 = 0,
                      const IEventParameter2 &parm2 = 0) const;
```

Condition: (3) AIX only

```
virtual IEventResult sendEvent(const IEvent &event) const;
```

Overrides IWindow::sendEvent (see page 938).

setBackgroundColor

Condition: (3) AIX only

```
IContainerControl &setBackgroundColor(const IColor &color);
```

Sets the background color.

Motif Information: This function is overridden under Motif so that the parent ScrolledWindow is also set to the specified color.

setCursor

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setCursor(
                            IContainerObject *object);
```

Gives the specified object cursor emphasis.

setDefaultAttribute

```
static void setDefaultAttribute(const Attribute &attribute);
```

Sets the default attribute for all subsequent containers.

attribute Use the attributes provided by IContainerControl::Attribute (see page 191) to specify the default attribute.

setDefaultStyle

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

Sets the default style for all subsequent containers.

style Use the styles provided by IContainerControl::Style (see page 204) to specify the default style.

setDeleteColumnsOnClose

```
virtual IContainerControl &setDeleteColumnsOnClose(
                            Boolean destroy = true);
```

Deletes all columns in the container when the container is deleted. Set only columns that are allocated by using operator new for automatic deletion. Because columns can be in multiple containers, the container's default behavior only removes columns but does not delete them when the container is deleted.

setDeleteObjectsOnClose

```
virtual IContainerControl &setDeleteObjectsOnClose(
                            Boolean destroy = true);
```

Deletes all objects in the container when the container is deleted. Because objects can be in multiple containers, the container's default behavior only removes objects but does not delete them when the container is deleted.

setDetailsViewSplit

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setDetailsViewSplit(
                            const IContainerColumn *lastColumnBeforeSplit,
                            unsigned long pixelsFromLeft = 50);
```

Splits the details view. You determine where the split occurs by specifying the last column to view in the left window. If a split bar already exists, it is moved to the new location. The *lastColumnBeforeSplit* parameter is the last column that will ever be scrolled in the left frame. The *pixelsFromLeft* parameter is the location of the split bar in pixels.

setEditColumn

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setEditColumn(
                            IContainerColumn *column);
```

Stores the specified column where editing occurs.

setEditMLE

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setEditMLE(
                            IMultiLineEdit *editField);
```

Stores the specified multiple-line edit (MLE) field being used for editing.

setEditObject

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setEditObject(
                            IContainerObject *object);
```

Stores the specified object where editing occurs.

setExtendedSelection

```
virtual IContainerControl &setExtendedSelection();
```

Sets the selection mode to extended selection. Extended selection is an enhanced version of single selection that lets the user select discontiguous sets of container items.

setIconSize

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setIconSize(
                            const ISize &iconSize);
```

Sets the icon or bit-map size for all objects.

setInUse

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setInUse(
                            IContainerObject *object,
                            Boolean inUse = true);
```

Gives the specified object in-use emphasis.

setLineSpacing

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setLineSpacing(long lineSpacing);
```

Sets the vertical distance between records. This is the amount you can add to the minimum distance required by the container control for emphasis painting. You cannot reduce the minimum distance required by the presentation system.

setMixedTargetEmphasis

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setMixedTargetEmphasis();
```

Sets the drag mode to mixed-target emphasis.

setMultipleSelection

```
virtual IContainerControl &setMultipleSelection();
```

Sets the selection mode to multiple selection. Multiple selection is a mode that gives the user unrestricted selection of objects.

setNormalTargetEmphasis

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setNormalTargetEmphasis();
```

Sets the drag mode to normal-target emphasis.

setOrderedTargetEmphasis

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setOrderedTargetEmphasis();
```

Sets the drag mode to ordered-target emphasis.

setRefreshOff

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setRefreshOff();
```

Disables refreshing the container until IContainerControl::refresh (see page 171) or setRefreshOn (see page 178) is called to enable refreshing it. This function sets the container's refresh state to off. After you set refresh to off in the container, you must call setRefreshOn followed by IContainerControl::refresh to show changes made while refresh was set to off.

setRefreshOn

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setRefreshOn(Boolean on = true);
```

Enables refreshing the container after changes. This function sets the container's refresh state to on. You should also call IContainerControl::refresh after this function to show changes made to the container while refresh was set to off.

setSelected

```
virtual IContainerControl &setSelected(
                            IContainerObject *object,
                            Boolean select = true);
```

Gives the specified object selection emphasis.

setSingleSelection

```
virtual IContainerControl &setSingleSelection();
```

Sets the selection mode to single selection. In this mode, the user (or your code) can only select a single object at a time. Selecting a new object removes the selection from a prior object. In a single-selection container one object is always selected. If you remove an object with selection emphasis, the container selects another object.

setTitle

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setTitle(const char *title);
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setTitle(
                            const IResourceId &resourceId);
```

Sets the title for the container that is displayed in all views.

setTitleAlignment

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setTitleAlignment(
                            TitleAlignment alignment = centered);
```

Sets the alignment attributes for the title. The default centers the title.

setTreeExpandIconSize

Condition: (1) Ignored on AIX

```
virtual IContainerControl &setTreeExpandIconSize(
                            const ISize &sizeIcon);
```

Changes the size of the expanded and collapsed icons.

setTreeItemIcons

```
virtual IContainerControl &setTreeItemIcons(
                            const IPointerHandle &expanded,
                            const IPointerHandle &collapsed);

virtual IContainerControl &setTreeItemIcons(
                            const IResourceId &expanded,
                            const IResourceId &collapsed);
```

Sets the expanded and collapsed icons for a tree view.

setTreeViewIndent

```
virtual IContainerControl &setTreeViewIndent(
                            long indentPixels = -1);
```

Sets the distance that children are offset horizontally from their parent.

showDetailsView

```
virtual IContainerControl &showDetailsView();
```

Sets the current view to details view. This member function displays the container objects as multiple columns of data with one row for each object. Optionally, you can divide the output into two regions using a single vertical split bar.

showDetailsViewTitles

```
virtual IContainerControl &showDetailsViewTitles(
                            Boolean show = true);
```

Displays column headings in details view. Although you set the column titles (that is, headings) for the individual column objects, you must use this function to tell the container control to display the column titles for the details view.

showFlowedNameView

```
virtual IContainerControl &showFlowedNameView();
```

Sets the current view to the flowed name view.

showFlowedTextView

```
virtual IContainerControl &showFlowedTextView();
```

Sets the current view to the flowed text view.

showIconView

```
virtual IContainerControl &showIconView();
```

Sets the current view to the icon view. If you have called arrangeIconView or have specified the autoPosition style, this member function displays the container objects as icons with the icon text below the icon. If you have more objects than will fit in a row, the objects flow into a second row. If there are more rows of objects than will fit in your container's window, the user can vertically scroll the container to see the rest of the objects.

Motif Information: In Motif, icon view always uses the style autoPosition. You cannot reposition the icons using the mouse. Functions that programmatically relocate icons, such as moveIconTo, are not supported.

showNameView

```
virtual IContainerControl &showNameView();
```

Sets the current view to the non-flowed name view. If there are more objects than will fit in the container, the user can vertically scroll the container to see the rest of the objects.

showObject

```
virtual IContainerControl &showObject(
                            IContainerObject *object,
                            Boolean visible = true);
```

Shows the specified object that is currently invisible or filtered from the container.

showSourceEmphasis

Condition: (1) Ignored on AIX

```
virtual IContainerControl &showSourceEmphasis(
                            IContainerObject *object,
                            Boolean source = true);
```

Condition: (1) Ignored on AIX

```
virtual IContainerControl &showSourceEmphasis(
                            Boolean source = true);
```

Draws source emphasis. If you specify an object, the emphasis is drawn around the object. Otherwise, the container itself gets the source emphasis.

showSplitBar

Condition: (1) Ignored on AIX

```
virtual IContainerControl &showSplitBar(
                            Boolean showSplitBar = true);
```

Adds the split bar to the details view work area.

showTextView

```
virtual IContainerControl &showTextView();
```

Sets the current view to the non-flowed text view.

Motif Information: Text view displays icons.

showTitle

Condition: (1) Ignored on AIX

```
virtual IContainerControl &showTitle(Boolean show = true);
```

Shows the container title.

showTitleSeparator

Condition: (1) Ignored on AIX

```
virtual IContainerControl &showTitleSeparator(
                            Boolean show = true);
```

Displays a horizontal separator between the title and the work area.

showTreeIconView

```
virtual IContainerControl &showTreeIconView();
```

Displays the objects as icons in a tree to represent their relationship to one another.

showTreeLine

```
virtual IContainerControl &showTreeLine(
                            long treeLinePixelWidth = -1);
```

Shows the lines connecting parents to children's records.

showTreeNameView

```
virtual IContainerControl &showTreeNameView();
```

Sets the current view to the tree name view.

showTreeTextView

```
virtual IContainerControl &showTreeTextView();
```

Displays the objects in a tree to represent their relationship to one another. Unlike showTreeIconView, this function only displays the object's text and not the object's icon.

Motif Information: Tree-text view displays icons.

sort

```
virtual IContainerControl &sort(
                            const IContainerControl::CompareFn &sortObject);
```

Sorts the container using a user-defined sort function, IContainerControl::CompareFn (see page 197).

sortByIconText

```
virtual IContainerControl &sortByIconText(
                              Boolean ascending = true);
```

Sorts the container by the icon's text.

splitBarOffset

Condition: (1) Ignored on AIX

```
unsigned long splitBarOffset() const;
```

Returns the offset of the split bar, in pixels, from the left side of the container window.

textRectangle

Condition: (1) Ignored on AIX

```
IRectangle textRectangle(
            const IContainerObject *object) const;
```

Returns the text rectangle in window coordinates.

title

Condition: (1) Ignored on AIX

```
virtual IString title() const;
```

Returns the title of the container.

titleRectangle

Condition: (1) Ignored on AIX

```
IRectangle titleRectangle() const;
```

Returns the rectangle that contains the container's title.

viewPortOnWindow

Condition: (1) Ignored on AIX

```
IRectangle viewPortOnWindow() const;
```

Retrieves the current work area in window coordinates.

viewPortOnWorkspace

Condition: (1) Ignored on AIX

```
IRectangle viewPortOnWorkspace() const;
```

Retrieves the current work area in workspace coordinates.

willDeleteColumnsOnClose

```
Boolean willDeleteColumnsOnClose() const;
```

If the columns will be deleted when the container is deleted, true is returned.

willDeleteObjectsOnClose

```
Boolean willDeleteObjectsOnClose() const;
```

If container objects will be deleted when the container is deleted, true is returned.

Figure 52. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow

Figure 52. Inherited Public Members

Member function	Class
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow

Figure 52. Inherited Public Members

Member function	Class
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow

Figure 52. Inherited Public Members

Member function	Class
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

attributes

```
unsigned long attributes(
                const IContainerObject *object) const;
```

Returns an attribute mask for the specified object.

baseRecordSize

```
virtual unsigned long baseRecordSize();
```

Returns the size of the RECORDCORE portion of the objects stored in a container.

calcMinimumSize

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

Returns a minimum size for the container.

column

```
IContainerColumn *column(
                    IContainerColumn *column,
                    unsigned long command,
                    Boolean visible);
```

Returns a column matching the command criteria.

containerAttributes

```
unsigned long containerAttributes() const;
```

Returns the current container attribute mask.

containerInfo

```
virtual void containerInfo() const;
```

Refreshes your copy of the container's state data.

containerKey

```
static IPrivateResource &containerKey();
```

Returns the key used to control access when updating the container. The key acts as a semaphore to the container updates.

containerList

```
static ICnrControlList &containerList();
```

Returns the list of containers in this application.

hwndAllocation

```
static IWindowHandle hwndAllocation();
```

Returns the container used for object allocation.

registerCallbacks

Condition: (3) AIX only

```
void 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.

setAttributes

```
virtual void setAttributes(
               IContainerObject *object,
               unsigned long attributeToTurnOn,
               unsigned long attributeToTurnOff);
```

Updates the attributes of the specified object.

setContainerAttributes

```
virtual void setContainerAttributes(
               unsigned long attributeToTurnOff,
               unsigned long attributeToTurnOn);
```

Sets the attributes into the container control.

setDefaultContainer

```
virtual void setDefaultContainer(
               const Attribute &attribute);
```

Initializes a container to the default styles.

setEmphasis

```
virtual void setEmphasis(
               IContainerObject *object,
               unsigned long emphasisAttribute,
               Boolean setOn = true);
```

Sets the specified object to the specified emphasis attribute.

topHandle

Condition: (3) AIX only

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

Returns the top, that is, the oldest ancestor, window system control of the window controls created by this object. Use the returned handle to show this object and destroy the window system controls during object destruction.

unregisterCallbacks

Condition: (3) AIX only

```
void unregisterCallbacks();
```

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

Figure 53. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow

Figure 53. Inherited Protected Members

Member function	Class
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow

Figure 53. Inherited Protected Members

Member function	Class
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

EnumerationOrder

```
public:
enum EnumerationOrder
  {
  itemOrder,
  zOrder,
  };
```

Use these enumerators to specify the order in which icons are enumerated:

itemOrder
Enumerate container records in item order; lowest to highest. Item order is similar to the order in which the records were added to the container, except you can explicitly insert records into the item order.

zOrder
Enumerate container records by z-order, first to last.

zOrder is the order in which icons appear in the icon view of the container: left to right, top to bottom, and then top icon to bottom icon as described above.

Note: This enumerator is only valid for the icon view.

TitleAlignment

```
public:
enum TitleAlignment
  {
  left,
  right,
  centered,
  };
```

Use these enumerators to specify the type of alignment for a title:

left
Left-justifies the container titles.

right
Right-justifies the container titles.

centered
Centers the container titles.

Chapter 38. IContainerControl::Attribute

Derivation

IBase
  IBitFlag
    IContainerControl::Attribute

Header File: icnrctl.hpp

The nested class IContainerControl::Attribute provides a set of valid container attributes for the members of the class IContainerControl (see page 149).

Attributes

alignTitleCentered
Centers the container title.

alignTitleLeft
Left-justifies the container title.

alignTitleRight
Right-justifies the container title.

classDefaultAttribute
Specifies the original default attribute for this class, which is orderedTargetEmphasis | detailsViewTitles | visibleTreeLine | readOnlyTitle | iconView.

detailsView
Shows the details view of the container.

detailsViewTitles
Shows the details view titles.

flowedView
Shows the flowed view of the container.

Note: This attribute is only valid when combined with the nameView attribute or the textView attribute.

handleDrawBackground
Causes draw-background messages to be dispatched to objects of classes derived from ICnrDrawHandler.

Note: The AIX release does not support this attribute because AIX does not support the ICnrDrawHandle.

handleDrawItem
Causes draw-item messages to be dispatched to objects of classes derived from ICnrDrawHandler.

© Copyright IBM Corp. 1994

Note: The AIX release does not support this attribute because AIX does not support the ICnrDrawHandle.

iconView
Shows the icon view of the container.

mixedTargetEmphasis
Sets the mixed-target emphasis.

nameView
Shows the name view of the container.

orderedTargetEmphasis
Sets the ordered-target emphasis.

readOnlyTitle
Disables the editing of the container title.

textView
Shows the text view of the container.

titleSeparator
Adds a separator line after the title.

treeView
Shows the tree view of the container.

visibleTitle
Shows the container title.

visibleTreeLine
Shows the tree line in the tree view.

Public Members

Figure 54. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 55. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 39. IContainerControl::ColumnCursor

Derivation

IBase
  IVBase
    IContainerControl::ColumnCursor

Header File: icnrctl.hpp

Objects of the IContainerControl::ColumnCursor class navigate through the container's details view columns.

Motif Information: The left-most column that the Container widget maintains automatically is never accessed by the ColumnCursor.

Constructors

```
public:
ColumnCursor(
   const IContainerControl &container,
   Boolean visibleOnly = false);
```

You can only construct objects of this class by using a cursor to iterate through all columns in the container meeting the visibility criteria (all columns or visible columns only). You must specify the following:

- The container where navigation is to occur
- Optionally, a Boolean indicating whether all columns, or only visible columns, are navigated

Public Members

current

```
virtual IContainerColumn *current() const;
```

Returns the current column of this cursor.

first

```
virtual IContainerColumn *first();
```

Points to and returns the first column in the container. If there are no objects in the container, 0 is returned. This function validates a cursor that was set invalid as a result of adding or removing columns from the container.

© Copyright IBM Corp. 1994

invalidate

```
virtual ColumnCursor &invalidate();
```

Flags the cursor as not valid.

isValid

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

Queries whether the cursor is valid.

The cursor is not valid if any of the following occur:

- You add a column after you created the cursor.
- You remove a column after you created the cursor.
- The last operation on the cursor put it into an undefined state.

For example, calling setToNext when the cursor is pointing to the last object in the container, causes the cursor to be not valid.

last

```
virtual IContainerColumn *last();
```

Points to and returns the last column in the container. Returns 0 if there are no columns in the container. This function validates a cursor that was set invalid as a result of adding or removing columns from the container.

next

```
virtual IContainerColumn *next();
```

Points to and returns the next or first column in the container. If the cursor has reached the end of the list of columns in the container, 0 is returned.

previous

```
virtual IContainerColumn *previous();
```

Points to and returns the previous or last column in container.

setCurrent

```
virtual ColumnCursor &setCurrent(
                        const IContainerColumn *currentColumn);
```

Sets the cursor to point to the specified column and validates the cursor. This function does not check for visibility.

setToFirst

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

Points to the first column in the container. If there are no objects in the container, false is returned. This function validates a cursor that was set invalid as a result of adding or removing columns from the container.

setToLast

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

Points to the last column in the container. If there is a last column, true is returned. Otherwise, false is returned.

setToNext

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

Points to the next column in the container. If the cursor does not point to a column, this function is the same as setToFirst. The cursor does not point to a column when it is initially created. In other words, setToNext on a new cursor is the same as setToFirst.

setToPrevious

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

Points to the previous or last column in the container.

Figure 56. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 40. IContainerControl::CompareFn

Derivation

IBase
  IVBase
    IContainerControl::CompareFn

Header File: icnrctl.hpp

Objects of the IContainerControl::CompareFn class provide a sort function for container objects. The container uses these objects to sort its contents. The class provides this sorting function by passing the compare function multiple sets of two objects. The compare function indicates whether the objects are equal or if one is before the other.

When the container calls IContainerControl::sort (see page 182), an IContainerControl::CompareFn object is passed as input to this function. Thereafter, the container calls IContainerControl::CompareFn::isEqual (see page 197) as needed to sort each container object.

Constructors

```
public:
CompareFn();
```

You can not construct objects of this class because it is an abstract base class.

Public Members

isEqual

```
virtual int isEqual(
              IContainerObject *object1,
              IContainerObject *object2,
              IContainerControl *container) const = 0;
```

Returns the result of an equivalency test between *object1* and *object2* in the specified container. The results indicate the following:

Less than 0 The first object (*object1*) is less than the second object (*object2*).

0 The first object is equal to the second object.

Greater than 0 The first object is greater than the second object.

Figure 57. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*

Figure 57. Inherited Public Members

Member function	Class
asString	IBase*
asString	IVBase*

© Copyright IBM Corp. 1994

Figure 57. Inherited Public Members

Member function	Class
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 41. IContainerControl::FilterFn

Derivation

IBase
  IVBase
    IContainerControl::FilterFn

Header File: icnrctl.hpp

Objects of the IContainerControl::FilterFn class provide a filter function for container objects. The container uses a FilterFn object to show a subset of the existing container objects. The class provides this sorting function based on a call to isEqual on the filter function.

When the container calls IContainerControl::filter (see page 160), an IContainerControl::FilterFn object is passed as input to this function. Thereafter, the container calls IContainerControl::FilterFn::isMemberOf (see page 149) as needed to filter each container object.

Constructors

```
public:
FilterFn();
```

You can not construct objects of this class because it is an abstract base class.

Public Members

isMemberOf

```
virtual Boolean isMemberOf(
                  IContainerObject *object,
                  IContainerControl *container) const = 0;
```

If the specified object is to remain in the container, true is returned. If the object is to be removed, false is returned.

Figure 58. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*

Figure 58. Inherited Public Members

Member function	Class
operator <<	IVBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 42. IContainerControl::Iterator

Derivation

IBase
  IVBase
    IContainerControl::Iterator

Header File: icnrctl.hpp

Objects of the IContainerControl::Iterator class apply a function to the container's objects. To perform a function on the objects of the container, derive a class from IContainerControl::Iterator and implement the behavior of IContainerControl::Iterator::applyTo (see page 200). Next, call IContainerControl::allObjectsDo (see page 151) with an object of this class. allObjectsDo calls applyTo on each object until one of the following occurs:

- IContainerControl::Iterator::applyTo returns false.
- IContainerControl::allObjectsDo has called applyTo for all the objects in the container.

Constructors

```
public:
Iterator();
```

You can not construct objects of this class because it is an abstract base class.

Public Members

applyTo

```
virtual Boolean applyTo(IContainerObject *object) = 0;
```

If the iteration is to continue, true is returned. If the iteration is to stop, false is returned.

Figure 59. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 43. IContainerControl::ObjectCursor

Derivation

IBase
  IVBase
    IContainerControl::ObjectCursor

Header File: icnrctl.hpp

Objects of the IContainerControl::ObjectCursor class iterate the container by finding objects based on the selection criteria established during construction of the cursor. The cursor is only valid until objects are added or removed from the container.

Constructors

```
public:
ObjectCursor(
   const IContainerControl &container,
   IContainerObject::Emphasis emphasis = IContainerObject::none,
   EnumerationOrder enumeration = itemOrder);

public:
ObjectCursor(
   const IContainerControl &container,
   const IContainerObject *parentObject,
   Boolean allDescendents = false);
```

You can construct objects of this class using one of the following:

- A cursor to iterate all records either:
  - In the container
  - Meeting the emphasis selection criteria

  Optionally, you can specify whether to iterate records in item order (the default) or z-order.
- A cursor to find all objects having the specified object as a parent.

Public Members

current

```
virtual IContainerObject *current() const;
```

Returns the current object of this cursor.

first

```
virtual IContainerObject *first();
```

© Copyright IBM Corp. 1994

Points to and returns the first object in the container that meets the criteria established during the construction of the cursor. If there are no objects in the container, 0 is returned. This function validates a cursor that was set invalid as a result of adding or removing columns from the container.

invalidate

```
virtual ObjectCursor &invalidate();
```

Flags the cursor as not valid.

isValid

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

Queries whether the cursor is valid. The cursor is not valid if any of the following occur:

- You add a column after you created the cursor.
- You remove a column after you created the cursor.
- The last operation on the cursor put it into an undefined state.

For example, calling setToNext when the cursor is pointing to the last object in the container, causes the cursor to be not valid.

last

```
virtual IContainerObject *last();
```

Points to and returns the last object in the container. Returns 0 if there are no objects in the container. This function validates a cursor that was set invalid as a result of adding or removing objects from the container.

next

```
virtual IContainerObject *next();
```

Points to and returns the next or first object in the container. If the cursor has reached the end of the list of objects in the container, 0 is returned.

previous

```
virtual IContainerObject *previous();
```

Points to and returns the previous or last object in the container.

setCurrent

```
virtual ObjectCursor &setCurrent(
                        const IContainerObject *currentObject);
```

Sets the cursor to point to the specified object and validates the cursor. This function does not check for visibility.

setToFirst

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

Points to the first object in the container. If there are no objects in the container, false is returned. This function validates a cursor that was set invalid as a result of adding or removing objects from the container.

setToLast

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

Points to the last object in the container. If there is a last object, true is returned. Otherwise, false is returned.

setToNext

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

Points to the next object in the container. If the cursor does not point to an object, this function is the same as setToFirst. The cursor does not point to an object when it is initially created. In other words, setToNext on a new cursor is the same as setToFirst.

setToPrevious

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

Points to the previous or last object in the container.

Figure 60. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 44. IContainerControl::Style

Derivation

IBase
  IBitFlag
    IContainerControl::Style

Header File: icnrctl.hpp

The nested class IContainerControl::Style provides a set of valid container styles for the members of the class IContainerControl (see page 149). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

autoPosition
Automatically positions objects added to the container in the icon view.

classDefaultStyle
Specifies the original default style for this class, which is singleSelection | IWindow::visible.

extendedSelection
Specifies that the container use extended selection. Extended selection is an enhanced version of single selection that lets the user select discontiguous sets of container items.

multipleSelection
Specifies that the container use multiple selection. Multiple selection is a mode that allows the user unrestricted selection of objects.

readOnly
Specifies that the text in the container cannot be edited. If this style is set, the user cannot edit anything in the container regardless of calls to the following:

- enableDataUpdate on the container object
- enableTitleUpdate on the container object
- enableDataUpdate on an IContainerColumn object
- enableHeadingUpdate on an IContainerColumn object

singleSelection
Specifies the container use single selection. In this mode, the user (or your code) can only select a single object at a time. Selecting a new object removes the selection from a prior object. In a single-selection container one object is

© Copyright IBM Corp. 1994

always selected. If you remove an object with selection emphasis, the container selects another object.

verifyPointers
Validates that container objects exist. If you specify this style, the presentation system checks that an object exists in the collection prior to using it. If a container record does not exist, an IAccessError exception is thrown.

Public Members

Figure 61. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 62. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 45. IContainerControl::TextCursor

Derivation

```
IBase
  IVBase
    IContainerControl::TextCursor
```

Header File: icnrctl.hpp

Objects of the IContainerControl::TextCursor class iterate through the container and find objects containing a specified text string. You specify this text string when creating the text cursor.

Constructors

```
public:
TextCursor(
   const IContainerControl &container,
   const char *text,
   Boolean isCaseSensitive = true,
   Boolean isFirstInRecord = false,
   Boolean isExactMatch = false);
```

You can construct objects of this class by specifying a text string to search the container for. Optionally, you can specify that the search return:

- A case sensitive match only
- A match only when the text is the first in the record
- An exact match of the text string only

When you construct a TextCursor, it is only valid for the objects shown in the currently displayed container view. For example, if you construct a TextCursor while the container is displaying the icon view and you want the application to iterate a TextCursor for children in a tree view, you must do the following:

- Switch the container to the tree view
- Create a new TextCursor

If you create a details view TextCursor, it searches all the text columns for the specified string.

Public Members

current

```
virtual IContainerObject *current() const;
```

Returns the current object of this cursor.

© Copyright IBM Corp. 1994

first

```
virtual IContainerObject *first();
```

Points to and returns the first object in the container with the matching text. If there are no objects in the container, 0 is returned. This function validates a cursor that was set invalid as a result of adding or removing columns from the container.

invalidate

```
virtual TextCursor &invalidate();
```

Flags the cursor as not valid.

isValid

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

Queries whether the cursor is pointing to a valid object. The cursor is not valid if any of the following occur:

- You add a column after you created the cursor.
- You remove a column after you created the cursor.
- The last operation on the cursor put it into an undefined state.

For example, calling setToNext when the cursor is pointing to the last object in the container, causes the cursor to be not valid.

last

```
virtual IContainerObject *last();
```

Points to an returns the last object in the container with matching text. Returns 0 if there are no object in the container. This function will validate a cursor that was set invalid as a result of adding or removing an object from the container.

next

```
virtual IContainerObject *next();
```

Points to and returns the next or first object in the container with the matching text. If the cursor has reached the end of the list of objects in the container, 0 is returned.

previous

```
virtual IContainerObject *previous();
```

Points to and returns the previous or last object in the container with the matching text.

setCurrent

```
virtual TextCursor &setCurrent(
                      const IContainerObject *currentObject);
```

Sets the cursor to point to the specified object.

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

Points to the first object in the container with the matching text. If there are no objects in the container, false is returned. This function validates a cursor that was set not valid as a result of adding or removing objects from the container.

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

Points to the last object in the container with the matching text. If there is a last object, true is returned. Otherwise, false is returned.

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

Points to the next or first object in the container with the matching text. If the cursor does not point to an object, this function is the same as setToFirst. The cursor does not point to an object when it is initially created. In other words, setToNext on a new cursor is the same as setToFirst.

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

Points to the previous or last object in the container.

Figure 63. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 46. IContainerObject

Derivation

IBase
  IVBase
    IContainerObject

Header File: icnrobj.hpp

IContainerObject is a required base class for all objects stored in a container. You must create all objects of this class using operator new, which is overloaded in this class to provide the storage allocation required by the underlying presentation system's container control.

You can use an IContainerObject object, as is, for all container views except the details view. For the details view, you must derive a class from IContainerObject to support additional data. If you do, the additional data must be the following:

- Contiguous with the data of this class
- One of the supported types defined by IContainerColumn (see page 135)

Note: You can only edit the string type. We recommend that you use the class IString to store text data in a derived object class.

We also recommend that you use a resource library to load icons because a resource library can do the following:

- Satisfying multiple requests for the same icon with a single load
- Manage multiple references to the same object, such that actual icon deletion occurs when the use count reaches 0

You can add IContainerObjects to multiple containers at the same time. When using functions in this class to modify an object's characteristics, the functions do so in all containers that hold the object. This is true only when the IContainerControl parameter for these functions is 0. IContainerControl (see page 149) provides functions that you can use to modify certain attributes of an object in a single container.

You cannot add the same container object twice in a container. OS/2 Presentation Manager keeps pointers to place the record in the container relative to other records. Presentation Manager does not let you add the same record twice. An IGUIErrorInfo is thrown if you try. You can create another object of this container object using the IContainer copy constructor and add that object to the container.

© Copyright IBM Corp. 1994

Constructors

```
public:
IContainerObject();

public:
IContainerObject(
   const IString &string,
   const IPointerHandle &iconHandle = 0);

public:
IContainerObject(
   const IString &string,
   const IResourceId &iconId);

public:
IContainerObject(
   const IResourceId &nameID,
   const IResourceId &iconId);

public:
IContainerObject(const IContainerObject &object);

public:
IContainerObject(
   const IString &string,
   unsigned long iconId);
```

You can construct objects of this class using one of the following:

- The default constructor which accepts no parameters.
- Providing an IString for text and an IPointerHandle (see page 657) for the icon.
- Providing an IString for text and an IResourceId (see page 736) for the icon.
- Providing resource IDs for both the text and the icon.
- Copying an existing IContainerObject or a class derived from IContainerObject. If you derive from IContainerObject, you should provide a copy constructor in the class to ensure that the object is copied correctly.
- Providing an IString for text and an unsigned long to use as a resource ID for the icon.

  Note: The User Interface Class Library provides this constructor to resolve a potential ambiguity since both IResourceIds and IPointerHandles can be built implicitly from an unsigned long integer.

Public Members

disableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerObject &disableDataUpdate(
                           IContainerControl *container = 0);
```

Disables user updates of this object.

disableDrop

Condition: (1) Ignored on AIX

```
virtual IContainerObject &disableDrop(
                            IContainerControl *container = 0);
```

Prevents any object from being dropped on this object.

enableDataUpdate

Condition: (1) Ignored on AIX

```
virtual IContainerObject &enableDataUpdate(
                            Boolean enable = true,
                            IContainerControl *container = 0);
```

Enables user updates of this object if the container in which this object resides is enabled for direct editing.

enableDrop

Condition: (1) Ignored on AIX

```
virtual IContainerObject &enableDrop(
                            Boolean enable = true,
                            IContainerControl *container = 0);
```

Allows other objects to be dropped on this object.

handleCursoredChange

Condition: (1) Ignored on AIX

```
virtual void handleCursoredChange(
               IContainerControl *container,
               Boolean acquired);
```

Called when cursored emphasis changes on an object and the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87). If the object has just acquired cursored emphasis, *acquired* is set to true.

handleInuseChange

Condition: (1) Ignored on AIX

```
virtual void handleInuseChange(
               IContainerControl *container,
               Boolean acquired);
```

Called when in-use emphasis changes on an object and the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87). If the object has just acquired in-use emphasis, *acquired* is set to true.

handleOpen

```
virtual void handleOpen(IContainerControl *container);
```

Called when the user either selects the **Enter** key when an object has the cursor or double-clicks the mouse select button on an object and the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87).

handleSelectedChange

```
virtual void handleSelectedChange(
              IContainerControl *container,
              Boolean acquired);
```

Called when selection emphasis changes on an object and the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87). If the object has just acquired selection emphasis, *acquired* is set to true.

handleTreeCollapse

```
virtual void handleTreeCollapse(
              IContainerControl *container);
```

Called when this object is a parent in the tree view and the descendent objects have been collapsed. This is only called when the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87).

handleTreeExpand

```
virtual void handleTreeExpand(IContainerControl *container);
```

Called when this object is a parent in the tree view and the descendent objects have been expanded. This is only called when the IContainerControl (see page 149) containing the object has an ICnrHandler (see page 87).

helpId

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

Called when an ICnrHelpEvent (see page 91) is dispatched to ICnrHandler (see page 87). By default, this function returns 0. If the ID returned is greater than 0, the default behavior of the container's help handler is to display the help panel.

You can override this function in your classes that inherit from IContainerObject to provide the ID of a help panel to display when help is requested for this object.

hide

```
virtual IContainerObject &hide(
                            IContainerControl *container = 0);
```

Makes this object invisible by filtering it out.

icon

```
virtual IPointerHandle icon() const;
```

Returns the icon handle of an object. If this object does not have an icon, the icon handle is zero.

iconOffset

Condition: (1) Ignored on AIX

```
static unsigned long iconOffset();
```

Returns the offset of the object's IPointerHandle (see page 657) for the icon. The offset lets you use the icon in a column of the details view.

Note: This function is obsolete. There is no replacement function.

iconText

```
virtual IString iconText() const;
```

Returns the text of an object.

iconTextOffset

Condition: (1) Ignored on AIX

```
static unsigned long iconTextOffset();
```

Returns the offset of the object's IString. The offset allows you to use the IString in a column of the details view.

Note: This function is obsolete. There is no replacement function.

isDropOnAble

Condition: (1) Ignored on AIX

```
virtual Boolean isDropOnAble(
                  IContainerControl *container = 0) const;
```

If this object can accept a drop, true is returned. If *container* is 0 and this object is shared amongst multiple containers, this function returns true *only* if the object can be dropped-on in all of the containers. If you specify a container, this function only queries this object's state in the specified container.

isInUse

Condition: (1) Ignored on AIX

```
virtual Boolean isInUse(
                  IContainerControl *container = 0) const;
```

If this object has in-use emphasis, true is returned. If *container* is 0 and this object is shared amongst multiple containers, this function returns true *only* if the object is in use in all of the containers. If you specify a container, this function only queries this object's state in the specified container.

isOpen

```
virtual Boolean isOpen() const;
```

If this object is open, true is returned.

isReadOnly

Condition: (1) Ignored on AIX

```
virtual Boolean isReadOnly(
                IContainerControl *container = 0) const;
```

If this object is read-only, true is returned. If *container* is 0 and this object is shared amongst multiple containers, this function returns true *only* if the object is read only in all of the containers. If you specify a container, this function only queries this object's state in the specified container.

Note: This function is obsolete. Use IContainerObject::isWriteable instead.

isRefreshOn

Condition: (1) Ignored on AIX

```
virtual Boolean isRefreshOn() const;
```

If this object is in a refreshable state, true is returned.

isVisible

```
virtual Boolean isVisible(
                IContainerControl *container = 0) const;
```

If this object is visible, true is returned. If *container* is 0 and this object is shared amongst multiple containers, this function returns true *only* if the object is visible in all of the containers. If you specify a container, this function only queries this object's state in the specified container.

isWriteable

Condition: (1) Ignored on AIX

```
virtual Boolean isWriteable(
                IContainerControl *container = 0) const;
```

Returns true if the data in the object can be edited.

objectCopy

```
virtual IContainerObject *objectCopy();
```

Called when it is necessary to make a copy of an object. This occurs when you call IContainerControl::copyObjectTo (see page 153).

operator ==

```
Boolean operator ==(const IContainerObject &object);
```

If the objects are at the same storage location, true is returned.

operator delete

```
void operator delete(void *);
```

Deallocates the storage of a class object.

operator new

```
void *operator new(size_t size);
```

Allocates storage for an object of this class.

refresh

Condition: (1) Ignored on AIX

```
virtual IContainerObject &refresh(
                            IContainerControl *container = 0);
```

Refreshes this object. The object's and container's refresh state must be set to on for this function to be valid. You can call IContainerObject::setRefreshOn (see page 216) to do so.

removeInUse

Condition: (1) Ignored on AIX

```
virtual IContainerObject &removeInUse(
                            IContainerControl *container = 0);
```

Removes the in-use emphasis for this object.

setClosed

```
virtual IContainerObject &setClosed();
```

Removes the open emphasis in all containers.

setIcon

```
virtual IContainerObject &setIcon(
                            const IResourceId &iconId);

virtual IContainerObject &setIcon(
                            const IPointerHandle &iconHandle);

virtual IContainerObject &setIcon(unsigned long iconId);
```

Sets a new icon for an object.

setIconText

```
virtual IContainerObject &setIconText(
                            const IString &iconText);

virtual IContainerObject &setIconText(const char *iconText);

virtual IContainerObject &setIconText(
                            const IResourceId &iconTextId);
```

Sets the new text for an object.

setInUse

Condition: (1) Ignored on AIX

```
virtual IContainerObject &setInUse(
                            Boolean inUse = true,
                            IContainerControl *container = 0);
```

Sets the in-use emphasis for this object.

setOpen

```
virtual IContainerObject &setOpen(Boolean open = true);
```

Sets the open emphasis.

setRefreshOff

Condition: (1) Ignored on AIX

```
virtual IContainerObject &setRefreshOff();
```

Prevents refreshing of this object.

setRefreshOn

Condition: (1) Ignored on AIX

```
virtual IContainerObject &setRefreshOn(
                            Boolean refreshOn = true);
```

Allows refreshing of this object.

show

```
virtual IContainerObject &show(
                            Boolean show = true,
                            IContainerControl *container = 0);
```

Makes this object visible.

Figure 64. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

baseRecord
```
IMiniCnrRecord *baseRecord();
```

Returns the address of the real base of this object.

decrementUseCount
```
virtual IContainerObject &decrementUseCount();
```

Decreases the count of containers having this object.

incrementUseCount
```
virtual IContainerObject &incrementUseCount();
```

Increases the count of containers having this object.

initialize
```
virtual IContainerObject &initialize();
```

Initializes an object after construction.

isAttribute
```
virtual Boolean isAttribute(
                  unsigned long attribute,
                  IContainerControl *container) const;
```

If the specified attribute is set to on, true is returned.

setAttributes
```
virtual IContainerObject &setAttributes(
                            unsigned long attributeTurnedOff,
                            unsigned long attributeTurnedOn,
                            IContainerControl *container = 0);
```

Sets some attributes to off while setting others to on. Use this function to toggle the state of mutually exclusive attributes. If you do not specify a container, the changes are made to all containers that have this object.

setBase
```
virtual IContainerObject &setBase(
                            const IMiniCnrRecord *baseRecord);
```

Stores the address of the real base of this object.

setEmphasis
```
virtual IContainerObject &setEmphasis(
                            unsigned long emphasisAttribute,
                            Boolean turnOn = true,
                            IContainerControl *container = 0);
```

Sets the specified emphasis attribute to on or off. If you do not specify a container, the changes are made to all containers that have this object.

useCount

```
unsigned long useCount() const;
```

Returns the number of containers having this object.

Enumerations

Emphasis

```
public:
enum Emphasis
  {
  none = 0,
  cursored = 1,
  inuse = 2,
  selected = 4,
  };
```

These enumerators specify a visible indication of the condition of an object affecting the ability of a user to interact with that object:

none
Provides no emphasis. Use this enumerator when no emphasis is required.

cursored
Provides a visible indication that the object contains the cursor.

inuse
Provides a visible indication that an object has been opened.

selected
Provides a visible indication that an object has been selected by the user.

Chapter 47. IContextHandle

Derivation

IBase
  IHandle
    IContextHandle

Header File: ihandle.hpp

Objects of the IContextHandle class access and manage the application context.

Motif Information: IContextHandle is an alias for the X Toolkit type XtAppContext.

Constructors

Condition: (3) AIX only

```
public:
IContextHandle(_XtAppStruct *value = 0);
```

You can construct objects of this class from an XtAppContext (a value of type IHandle::Value), which defaults to 0.

Public Members

operator _XtAppStruct *

Condition: (3) AIX only

```
operator _XtAppStruct *() const;
```

Returns the IContextHandle value.

Figure 65. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 48. IControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl

Header File: icontrol.hpp

The IControl class is the abstract base class for classes representing window system controls such as list boxes, entry fields, and push buttons.

Portability Considerations: AIX ignores the styles group and tabStop. The library uses the default Motif tab group behavior instead, to maintain the Motif look and feel.

Nested Classes

IControl::Style

Constructors

```
protected:
IControl();
```

You cannot construct objects of this class because it is an abstract base class.

Public Members

disableGroup

Condition: (1) Ignored on AIX

```
virtual IControl &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

© Copyright IBM Corp. 1994

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.

Motif Information: 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 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

Condition: (1) Ignored on AIX

```
virtual IControl &disableTabStop();
```

Removes the tabStop style from a control, therefore preventing this control from being tabbed to.

Motif Information: AIX ignores the tabStop style.

enableGroup

Condition: (1) Ignored on AIX

```
virtual IControl &enableGroup(Boolean enable = true);
```

Sets the group style of a control, therefore making this control first in a cursorable group of controls.

Motif Information: AIX ignores the group style.

enableTabStop

Condition: (1) Ignored on AIX

```
virtual IControl &enableTabStop(Boolean enable = true);
```

Sets the tabStop style of a control, therefore identifying this control as one that can be tabbed to.

Motif Information: AIX ignores the tabStop style.

isGroup

Condition: (1) Ignored on AIX

```
virtual Boolean isGroup() const;
```

If the control has the group style set, true is returned. Otherwise, false is returned.

Motif Information: AIX ignores the group style, therefore this function returns false.

isTabStop

Condition: (1) Ignored on AIX

```
virtual Boolean isTabStop() const;
```

If the control has the tabStop style set, true is returned. Otherwise, false is returned.

Motif Information: AIX ignores the tabStop style, therefore this function returns false.

Figure 66. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow

Figure 66. Inherited Public Members

Member function	Class
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow

Figure 66. Inherited Public Members

Member function	Class
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow

Figure 66. Inherited Public Members

Member function	Class
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 67. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 49. IControl::Style

Derivation

IBase
  IBitFlag
    IControl::Style

Header File: icontrol.hpp

The nested class IControl::Style provides a set of valid control styles for the static member functions of the class IControl (see page 220). You can use these styles in conjunction with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

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.

tabStop
Identifies the control as one the user can tab to.

Portability Considerations: 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.

Public Members

Figure 68. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 69. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 50. IControlEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent

Header File: ictlevt.hpp

Objects of the IControlEvent class represent a notification event from a control. Many control event handler classes, such as IEditHandler (see page 247), 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.

Motif Information: An IControlEvent encapsulates various widget callbacks and certain X events.

Presentation Manager Information: An IControlEvent encapsulates the Presentation Manager message WM_CONTROL.

Constructors

```
public:
IControlEvent(const IEvent &event);

public:
IControlEvent(const IControlEvent &event);
```

Although you can construct objects of this class, typically IHandler::dispatchHandlerEvent (see page 404) creates objects of this class from an object of the class IEvent (see page 275) or another IControlEvent object.

Public Members

controlId

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

Returns the window ID of the control the event applies to.

controlWindow

```
virtual IWindow *controlWindow() const;
```

Returns a pointer to the control object the event applies to. The return value can be 0.

© Copyright IBM Corp. 1994

Figure 70. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 51. ICritSec

Derivation

IBase
  ICritSec

Header File: icritsec.hpp

You can declare instances of the ICritSec class within 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.

Instances 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 declare an instance 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 an ICritSec, use semaphores as implemented by the class IResourceLock (see page 744).

The following is an example of using ICritSec:

```
{
ICritSec lock;
// Manipulate static data without interruption...
}
```

Motif Information: 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 prior to creation of the object.

Because signals are masked, some services may not operate correctly inside a critical section. An example of this is timer signals. If you attempt to lock an ISharedResource inside a critical section, any timeout you specify will be ineffective because the timer signal is masked.

Constructors

```
public:
ICritSec();
```

© Copyright IBM Corp. 1994

All the function of this class is accomplished through its constructor and destructor. The constructor causes a critical section to be entered. The corresponding destructor exits the critical section.

Public Members

Figure 71. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 52. ICurrentApplication

Derivation

IBase
  IVBase
    IApplication
      ICurrentApplication

Header File: iapp.hpp

Objects of the ICurrentApplication class represent the program that is currently running. There is a single object of this class. You can obtain a reference to it by using the static member IApplication::current (see page 29). The object of this class contains information that the User Interface Class Library maintains for each running application.

Constructors

```
protected:
ICurrentApplication();
```

The constructor for this class is protected to ensure the static member IApplication::current (see page 29) points to the only instance.

Public Members

argc
```
virtual int argc() const;
```

Obtains the number of application arguments.

argv
```
virtual IString argv(int argumentNumber) const;
```

Obtains the specified argument to the application.

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

Returns a diagnostic representation of the object.

exit
```
virtual ICurrentApplication &exit();
```

© Copyright IBM Corp. 1994

Ends the current thread of execution. If the current thread is thread 1, the process is ended.

resourceLibrary

```
virtual IResourceLibrary &resourceLibrary() const;
```

Obtains a reference to the User Interface Class Library resource library from which resources required by the base library code are loaded.

Presentation Manager Information: If you call ICurrentApplication::setResourceLibrary (see page 232) 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 the environment variable has not been set, the default (dde4u001.dll) is used.

run

```
virtual ICurrentApplication &run();
```

Initiates the processing of events in the current process.

setArgs

```
virtual ICurrentApplication &setArgs(
                              int argc,
                              const char * const argv);
```

Sets the program arguments. Call this member function from your application's main function, passing its *argc* and *argv* argument values.

setResourceLibrary

```
virtual ICurrentApplication &setResourceLibrary(
                              const char *resLibName);
```

Sets the resource library from which the class library resources are loaded.

Motif Information: The AIX environment uses the standard X resource database, therefore this function has no effect.

Presentation Manager Information: 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 .exe.

setUserResourceLibrary

```
virtual ICurrentApplication &setUserResourceLibrary(
                              const char *resLibName);
```

Sets the user resource library from which application resources are obtained.

Motif Information: The AIX environment uses the standard X resource database, therefore this function has no effect.

Presentation Manager Information: The default is IResourceId (see page 736). If you specify 0 for *resLibName*, the application should use the .exe.

userResourceLibrary

```
virtual IResourceLibrary &userResourceLibrary() const;
```

Obtains a reference to the default user resource library for this application. The default is the .exe itself. If you do not explicitly specify a resource library, IResourceId (see page 736) uses this resource library.

Figure 72. Inherited Public Members

Member function	Class
adjustPriority	IApplication
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IApplication
asString	IBase*
asString	IVBase*
asString	IApplication
id	IApplication
operator <<	IBase*
operator <<	IVBase*
setPriority	IApplication

Note: * Application Support Class Library

Protected Members

pib

Condition: (2) OS/2 Only

```
struct pib_s &pib();
```

Returns a pointer to the current process's process information block.

Figure 73. Inherited Protected Members

Member function	Class
setId	IApplication

Note: * Application Support Class Library

Chapter 53. ICurrentThread

Derivation

IBase
  IVBase
    IThread
      ICurrentThread

Header File: ithread.hpp

The ICurrentThread class provides additional functions that can only be applied to the current thread of execution. You can only obtain a reference to the sole instance of this class by using the static member function IThread::current (see page 892).

Constructors

```
protected:
ICurrentThread();
```

The User Interface Class Library protects the constructor for this class in order to prevent the creation of objects of this class except from within IThread::current (see page 892).

Public Members

anchorBlock

Condition: (2) OS/2 Only

```
virtual IAnchorBlockHandle anchorBlock() const;
```

Obtains the anchor block handle for the thread. If the anchor block handle is not initialized, it is 0.

appContext

Condition: (3) AIX only

```
virtual IContextHandle appContext() const;
```

Obtains the application context handle for the thread. If the application context handle is not initialized, it is 0.

appShell

Condition: (3) AIX only

```
virtual IWindowHandle appShell() const;
```


© Copyright IBM Corp. 1994

Obtains the application shell handle for the thread. If the application shell handle is not initialized, it is 0.

exit

```
virtual void exit(unsigned long returnCode);
```

Terminates the current thread with the specified return value.

id

```
virtual IThreadId id() const;
```

Returns the current thread's ID.

Motif Information: The AIX release of the User Interface Class Library supports only one thread. Therefore, this function returns the value 1.

initializeGUI

```
virtual void initializeGUI(long queueSize = 0);
```

Initializes the graphical user interface (GUI).

Motif Information: Initializes the X library enviroment by calling XtAppInitialize establishing a connection to the display. This function also creates an application context and an initial application shell.

Presentation Manager Information: Sets up a Presentation Manager environment (anchor block and message queue).

initializePM

Condition: (2) OS/2 Only

```
virtual void initializePM(long queueSize = 0);
```

Initializes the OS/2 Presentation Manager.

Note: This function is obsolete. Use ICurrentThread::initializeGUI instead.

isGUIInitialized

```
virtual Boolean isGUIInitialized() const;
```

If the GUI environment is active for this thread, true is returned.

isPMInitialized

Condition: (2) OS/2 Only

```
virtual Boolean isPMInitialized() const;
```

If the OS/2 Presentation Manager environment is active for this thread, true is returned.

Note: This function is obsolete. Use ICurrentThread::isGUIInitialized instead.

isTopLevelShell

Condition: (3) AIX only

```
virtual Boolean isTopLevelShell() const;
```

If the application shell window is completely initialized, true is returned.

isXerrorCodeAvailable

Condition: (3) AIX only

```
virtual Boolean isXerrorCodeAvailable() const;
```

If an X library error code is available for querying, true is returned. This function uses XerrorCode to retrieve the error code, which resets this flag.

messageQueue

Condition: (2) OS/2 Only

```
virtual IMessageQueueHandle messageQueue();
```

Condition: (2) OS/2 Only

```
virtual IMessageQueueHandle messageQueue() const;
```

Obtains the message queue handle for the thread. If the message queue handle is not initialized, it is 0.

processMsgs

```
virtual void processMsgs();
```

Gets messages from the message queue and dispatches the messages to the appropriate handlers.

Motif Information: ICurrentThread dispatches messages until it receives the client message WM_QUIT or the application is terminated through the Motif Window Manager.

Presentation Manager Information: Messages are dispatched until the WM_QUIT message is received.

remainingStack

Condition: (1) Ignored on AIX

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

Returns an approximation of the number of stack bytes remaining. If the stack is not initialized, it is 0.

setTopLevelShell

Condition: (3) AIX only

```
virtual void setTopLevelShell() const;
```

Called when the application shell window has been initialized. A user of the User Interface Class Library should never call this member function.

setXerrorCode

Condition: (3) AIX only

```
virtual void setXerrorCode(int errorCode);
```

Sets the error code that the last call to an X library function reported. Typically, you use 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. A user of the User Interface Class Library should never call this member function.

sleep

```
virtual ICurrentThread &sleep(unsigned long milliSeconds);
```

Suspends the thread for a specified number of milliseconds.

Motif Information: 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.

suspend

Condition: (1) Ignored on AIX

```
virtual void suspend();
```

Checks to see that the thread is not suspended if it is controlling the GUI. Otherwise, the application hangs.

Motif Information: You cannot suspend or resume AIX threads because of the User Interface Class Library (AIX release) single-thread limitation.

terminateGUI

```
virtual void terminateGUI();
```

Shuts down the GUI environment. If you start a thread with IThread::autoInitGUI (see page 892) returning true, this function is called automatically by the User Interface Class Library once IThreadFn::run (see page 902) completes. If you do the following, the library does not call terminateGUI and it is your application's responsibility to do so:

- Start the thread with autoInitGUI returning false
- Stop the thread using IThread::stop (see page 898)
- Stop the thread using ICurrentThread::exit (see page 235)

If you do not call this function, you can tie up system resources until your application ends.

Presentation Manager Information: Before calling WinTerminate, you must destroy all windows and the message queue. This function destroys the message queue, but it is still the application's responsibility to destroy (that is, delete) all of the window objects for the thread. If you do not, the return value of WinTerminate and any subsequent calls is indeterminate.

terminatePM

Condition: (2) OS/2 Only

```
virtual void terminatePM();
```

Shuts down the OS/2 Presentation Manager environment.

Note: This function is obsolete. Use ICurrentThread::terminateGUI instead.

waitFor

Condition: (1) Ignored on AIX

```
virtual ICurrentThread &waitFor(
                        const IThread &anotherThread);
```

Causes the current thread to wait for a specified thread to terminate.

Motif Information: 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.

waitForAllThreads

Condition: (1) Ignored on AIX

```
virtual ICurrentThread &waitForAllThreads();
```

Causes the current thread to wait for all secondary threads to terminate.

Motif Information: 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.

waitForAnyThread

Condition: (1) Ignored on AIX

```
virtual IThreadId waitForAnyThread();
```

Causes the current thread to wait for the first termination to occur of any other threads in the current process. Returns the terminated thread's ID.

Motif Information: 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.

XerrorCode

Condition: (3) AIX only

```
virtual int XerrorCode() const;
```

Returns the X library error code set by the member function setXerrorCode. You can retrieve the error code until a new X library error code is set by setXerrorCode, but the flag indicating 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. A user of the User Interface Class Library should never have to call this member function.

Figure 74. Inherited Public Members

Member function	Class
adjustPriority	IThread
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IThread
asString	IBase*
asString	IVBase*
asString	IThread
autoInitGUI	IThread
autoInitPM	IThread
id	IThread
isStarted	IThread
messageQueue	IThread
operator <<	IBase*
operator <<	IVBase*
priorityClass	IThread
priorityLevel	IThread
queueSize	IThread
relatedHandlesList	IThread
resume	IThread
setAutoInitGUI	IThread
setAutoInitPM	IThread
setPriority	IThread
setQueueSize	IThread
setRelatedHandlesList	IThread
setStackSize	IThread
setVariable	IThread

Figure 74. Inherited Public Members

Member function	Class
setWindowList	IThread
stackSize	IThread
start	IThread
stop	IThread
stopProcessingMsgs	IThread
suspend	IThread
variable	IThread
windowList	IThread

Note: * Application Support Class Library

Protected Members

startedThread

```
virtual IStartedThread *startedThread() const;
```

Returns a pointer to IStartedThread for the current thread.

Figure 75. Inherited Protected Members

Member function	Class
operator =	IThread
startedThread	IThread

Note: * Application Support Class Library

Chapter 54. IDeviceColor

Derivation

IBase
  IVBase
    IColor
      IDeviceColor

Header File: icolor.hpp

Use the IDeviceColor class to work with device-independent color indexes.

Motif Information: The color white is used for IDeviceColor::background. The color black is used for IDeviceColor::neutral and IDeviceColor::defaultColor.

Constructors

```
public:
IDeviceColor(DeviceColor);
```

You can only construct objects of this class from a DeviceColor (see page 242) enumerator. The internal device color is set using the specified *DeviceColor*.

Public Members

deviceColor

```
DeviceColor deviceColor() const;
```

Returns the enumeration DeviceColor (see page 242) used to create this object.

Figure 76. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asPixel	IColor
asRGBLong	IColor
asString	IBase*
asString	IVBase*
blueMix	IColor
greenMix	IColor
index	IColor
operator !=	IColor
operator <<	IBase*

Figure 76. Inherited Public Members

Member function	Class
operator <<	IVBase*
operator =	IColor
operator ==	IColor
redMix	IColor
setBlue	IColor
setGreen	IColor
setRed	IColor
value	IColor

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Enumerations

DeviceColor

```
public:
enum DeviceColor
  {
  defaultColor,
  background,
  neutral,
  };
```

Use these enumerators to specify the device-independent color of the object you are constructing:

defaultColor
Specifies the default color.

background
Specifies the background color.

neutral
Specifies the neutral color.

Motif Information: The color values in Motif are hard coded. The background color is white, the neutral color is black, and the defaultColor is black.

Chapter 55. IDisplayHandle

Derivation

IBase
  IHandle
    IDisplayHandle

Header File: ihandle.hpp

Objects of the IDisplayHandle class wrapper X Toolkit objects of type Display *.

Motif Information: IDisplayHandle is an alias for a pointer to the X Library type Display.

Constructors

Condition: (3) AIX only

```
public:
IDisplayHandle(_XDisplay *value = 0);
```

You can construct objects of this class from an _XDisplay (a value of type IHandle::Value), which defaults to 0.

Public Members

operator _XDisplay *

Condition: (3) AIX only

```
operator _XDisplay *() const;
```

Returns the IDisplayHandle value.

Figure 77. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 56. IDynamicLinkLibrary

Derivation

IBase
  IVBase
    IResourceLibrary
      IDynamicLinkLibrary

Header File: ireslib.hpp

The IDynamicLinkLibrary class provides the support for loading OS/2 resources from a dynamic link library (.dll).

Motif Information: The AIX release of the User Interface Class Library does not support this class.

Constructors

Condition: (2) OS/2 Only

```
public:
IDynamicLinkLibrary(const char *resourceFileName);
```

Condition: (2) OS/2 Only

```
public:
IDynamicLinkLibrary(const IModuleHandle &moduleHandle);
```

Condition: (2) OS/2 Only

```
public:
IDynamicLinkLibrary(const IDynamicLinkLibrary &dllLibrary);
```

You can construct objects of this class in the following ways:

- By providing a name to use for the dynamic link library. If you want to search the LIBPATH for the .dll, specify *resourceFileName* without the path or extension.
- From an IModuleHandle (see page 534) obtained by loading the .dll explicitly.
- By copying an existing IDynamicLinkLibary.

Public Members

asDebugInfo

Condition: (2) OS/2 Only

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

Overridden to provide debugging information about the class object.

© Copyright IBM Corp. 1994

asString

Condition: (2) OS/2 Only

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

Overridden to provide textual information about the class object.

close

Condition: (2) OS/2 Only

```
virtual IDynamicLinkLibrary &close();
```

Closes a dynamic link library.

fileName

Condition: (2) OS/2 Only

```
virtual IString fileName() const;
```

Overridden to return the name of the dynamic link library.

handle

Condition: (2) OS/2 Only

```
virtual IModuleHandle handle() const;
```

Overridden to provide the dynamic link module handle.

isEntryPoint32Bit

Condition: (2) OS/2 Only

```
Boolean isEntryPoint32Bit(
          unsigned long procedureOrdinal) const;
```

Condition: (2) OS/2 Only

```
Boolean isEntryPoint32Bit(const char *procedureName) const;
```

If the entry point with the specified name or ordinal is a 32-bit function, true is returned.

isOpen

Condition: (2) OS/2 Only

```
virtual Boolean isOpen() const;
```

Overridden to return the open state of the dynamic link library.

open

Condition: (2) OS/2 Only

```
virtual IDynamicLinkLibrary &open();
```

Opens a dynamic link library.

operator =

Condition: (2) OS/2 Only

```
IDynamicLinkLibrary &operator =(
                      const IDynamicLinkLibrary &dllLibrary);
```

Assigns one dynamic link library to another.

procAddress

Condition: (2) OS/2 Only

```
void *procAddress(const char *procedureName) const;
```

Condition: (2) OS/2 Only

```
void *procAddress(unsigned long procedureOrdinal) const;
```

Loads the specified procedure address from a dynamic link library.

Figure 78. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IResourceLibrary
asString	IBase*
asString	IVBase*
asString	IResourceLibrary
fileName	IResourceLibrary
handle	IResourceLibrary
isOpen	IResourceLibrary
loadAccelTable	IResourceLibrary
loadBitmap	IResourceLibrary
loadDialog	IResourceLibrary
loadHelpTable	IResourceLibrary
loadIcon	IResourceLibrary
loadMenu	IResourceLibrary
loadMessage	IResourceLibrary
loadPointer	IResourceLibrary
loadString	IResourceLibrary
operator <<	IBase*
operator <<	IVBase*
operator =	IResourceLibrary
tryToLoadBitmap	IResourceLibrary
tryToLoadIcon	IResourceLibrary
tryToLoadMessage	IResourceLibrary
tryToLoadString	IResourceLibrary

Note: * Application Support Class Library

Chapter 57. IEditHandler

Derivation

IBase
  IVBase
    IHandler
      IEditHandler

Header File: iedithdr.hpp

Objects of the IEditHandler class handle 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:

- IComboBox (see page 108)
- IEntryField (see page 256)
- IMultiLineEdit (see page 557)
- IProgressIndicator (see page 679)
- ISlider (see page 792)
- ISpinButton (see page 803)

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 (see page 403) 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 (see page 227) object and routes that object to the IEditHandler::edit (see page 248) 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 specifies whether the control event is passed on for additional processing:

true The edit event requires no additional processing. Do not pass it to another handler.

false Pass the edit event to the next handler for additional processing, 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 (see page 947) to dispatch the edit event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (see page 946) to process the edit event.

© Copyright IBM Corp. 1994

Constructors

```
protected:
IEditHandler();
```

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.

Public Members

Figure 79. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If an edit event is received, the appropriate virtual function is called.

edit

```
virtual Boolean edit(IControlEvent &event) = 0;
```

Implemented by the derived classes to process input change events.

Figure 80. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 58. IEditVerifyEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IEditVerifyEvent

Header File: iedvevt.hpp

Objects of the IEditVerifyEvent class represent a pending change to the text data in an editable text control.

Changes to the text might include the following:

- Insertion
- Replacement
- Deletion

The change can be via keyboard entry, cut and paste, or using the text control's functions. Depending on how this event is processed, the change will be:

- Successful (either exactly as proposed, or you can modify the proposed text data)
- Unsuccessful

If successful, a control event is sent to the IEditHandler signaling that the change has occurred.

An IEditVerifyHandler constructs and processes IEditVerifyEvents. They can be constructed elsewhere as well. You must write your own handler class derived from IEditVerifyHandler. You can then use this handler to gain access to IEditVerifyEvents, allowing you to edit and/or verify proposed changes to an editable text control.

IEditVerifyEvents are first dispatched to the handlers attached to the control itself, then to the handlers attached to the owner window of the control.

Portability Considerations: This class is not yet available in the OS/2 version.

Constructors

```
public:
IEditVerifyEvent(const IEvent &event);

public:
IEditVerifyEvent(const IEditVerifyEvent &event);
```

Although you can construct objects of this class, typically IEditVerifyHandler::dispatchHandlerEvent (see page 255) creates objects of this class

© Copyright IBM Corp. 1994

from an object of the class IEvent (see page 275) or the class IEditVerifyEvent (see page 249) in reponse to the user changing text.

Public Members

allowChange

```
IEditVerifyEvent &allowChange(Boolean allow = true);
```

Sets the disposition of the proposed change.

allow The proposed change can:

- Succeed, subject to any changes made in the data of this event (true)
- Not succeed (false)

endPosition

```
unsigned long endPosition() const;
```

Returns the position of the last character in the control's current data to be affected by the proposed change. If the proposed change is an insert, both startPosition (see page 251) and endPosition (see page 250) will have the same value; this value is the point of insertion.

isChangeAllowed

```
Boolean isChangeAllowed() const;
```

Returns whether the proposed change is allowed to:

- Succeed, subject to any changes made in the data of this event (true)
- Not succeed (false)

proposedText

```
IString proposedText() const;
```

Returns a IString object containing the proposed text to be inserted or overwrittten into the control's data. This is a copy of the data in this event; if you want to modify the proposed text, you must call setProposedText (see page 251) with a IString object containing your modified text. The IString object can be the one returned by proposedText, or another one.

If proposedText is a null IString, the proposed change is a deletion from startPosition (see page 251) to endPosition (see page 250).

resultingText

```
IString resultingText() const;
```

Returns a IString object containing the result of combining the control's text with the proposed text; this would be the control's data if the change were allowed to occur. Typically, this text is examined to see if the proposed change would result in valid overall contents for the control if the change were allowed.

After examining it, if you want to change text to make it valid, work with the proposed text only, not the resulting text. Passing the resulting text on a call to setProposedText (see page 251) would not only add the proposed text but would add a second copy of the control's text. This is usually not what you want to do.

setEndPosition

```
IEditVerifyEvent &setEndPosition(unsigned long endPos);
```

Sets the position of the last character in the control's current data affected by the proposed change. If the proposed change is an insert, set both startPosition (see page 251) and endPosition (see page 250) to the same value; this value is the point of insertion.

setProposedText

```
IEditVerifyEvent &setProposedText(IString text);
```

Sets a new value for the pending text to be inserted or overwritten into the control's data. The *text* parameter can be the object returned by proposedText (see page 250) with your modifications, or it can be a completely different IString object. The IString object can be longer or shorter than the original text.

setStartPosition

```
IEditVerifyEvent &setStartPosition(unsigned long startPos);
```

Sets the position of the first character in the control's current data affected by the pending change. If the change is an insert, you should set both startPosition (see page 251) and endPosition (see page 250) to the same value; this value is the point of insertion.

startPosition

```
unsigned long startPosition() const;
```

Returns the position of the first character in the control's current data which will be affected by the pending change. If the change is an insert, both startPosition (see page 251) and endPosition (see page 250) will have the same value; this value is the point of insertion.

Figure 81. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 59. IEditVerifyHandler

Derivation

IBase
  IVBase
    IHandler
      IEditVerifyHandler

Header File: iedvhdr.hpp

Objects of the IEditVerifyHandler class process edit (change) verify events for the following controls:

- IComboBox (see page 108)
- IEntryField (see page 256)
- IMultiLineEdit (see page 557)
- ISpinButton (see page 803)

Edit-verify events describe a proposed change to the data in an editable text control. Such events result from a user attempting to change an input text value, by one of the following actions:

- Typing into a control
- Cutting or pasting into a control
- Calling setText for a control
- Clicking the button on a spin button
- Selecting from a combination box's list

In short, you can verify any change to the text data of these controls in this handler.

This handler is called after IKeyboardHandlers, but before IEditHandlers and before any of the changes in the text values show up on the screen.

You can create a handler derived from IEditVerifyHandler and attach it to either the control whose input users can change or to the control's owner window. You do this by calling IHandler::handleEventsFor (see page 403) to pass the appropriate control window or owner window to the edit-verify handler.

When the edit-verify handler receives an edit-verify event, it creates an IEditVerifyEvent (see page 249) object and routes that object to the editVerify virtual function. You must override this virtual function to supply your own specialized processing of a edit-verify event. In your processing, you decide whether the proposed change is:

- Successful (either exactly as proposed, or you can modify the proposed text data)
- Unsuccessful

The IEditVerifyEvent, which is routed to the virtual function, contains both the proposed data and the allowChange flag. This is the event you modify in your processing.

When processing, you can do the following:

© Copyright IBM Corp. 1994

- Increase the size of the proposed text
- Decrease the size of the proposed text
- Change the value of of the proposed text

Handlers which are called after yours can set the allowChange flag to false. After all the handlers have been called, the final state of allowChange determines whether a control event is created. The default behavior is successful (true). The control event is sent to IEditHandler signaling the change to the control.

The return value from the virtual function (which is not the same thing as allowChange) specifies whether the control event is passed on for additional processing, as follows:

true The edit-verify event requires no additional processing. Do not pass it to another handler.

false Pass the edit-verify event to the next handler for additional processing, as follows:

- If there is another handler for the control or window, pass the edit-verify event to the next handler.
- If this is the last handler for the control, call IWindow::dispatch (see page 947) to dispatch the edit-verify event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (see page 946) to process the edit-verify event.

Note: Additional processing by another handler can alter the proposed data again, and as possibly modify the allowChange flag.

Portability Considerations: When possible, use this class instead of IKeyboardHandler to avoid some portability problems with that class.

Presentation Manager Information: The OS/2 release of the User Interface Class Library does not yet support this class.

Constructors

```
protected:
IEditVerifyHandler();
```

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.

Public Members

Figure 82. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

IWindow (see page 924) calls this function when a window event occurs. If this function returns true, the event is not processed by additional handlers. In such cases, a result must be placed in the specified IEvent (see page 275) object.

editVerify

```
virtual Boolean editVerify(IEditVerifyEvent &event) = 0;
```

Implemented by derived classes to process an edit-verify event. If you do not override this function, you can not create a derived class.

Figure 83. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 60. IEntryField

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IEntryField
```

Header File: ientryfd.hpp

Objects of the IEntryField class create and manage entry field controls, known as text widgets. You can implement specific keystroke processing, such as creating a numeric-only input field, by deriving a handler from IEditVerifyHandler (see page 253) and attaching your handler to the entry field object.

You derive classes from the following handlers and attach them to an entry field control:

- IEditHandler (see page 247)
- IEditVerifyHandler (see page 253)
- IFocusHandler (see page 307)
- IPaintHandler (see page 647)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IResizeHandler (see page 732)
- IScrollHandler (see page 760)

Motif Information: The User Interface Class Library implements this class with the XmText widget to allow more uniformity between IEntryField and IMultiLineEdit. Additionally, XmTextField does not permit a proper implementation of this class.

Nested Classes

IEntryField::Style

Constructors

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

public:
IEntryField(unsigned long id, IWindow *parent);
```


© Copyright IBM Corp. 1994

```
public:
IEntryField(const IWindowHandle &handle);

protected:
IEntryField();
```

You can construct objects of this class by:

- Using the parent window, owner window, optional size and location, and optional style parameters
- Using the parent window
- Using the handle of an existing entry field
- Using a protected constructor that is used in the implementation of IComboBox

id An entry field control ID.

parent The parent window.

owner The owner window.

initial (Optional) A rectangle for the control. It specifies the initial position and size of the object you are constructing.

style (Optional) The initial style for the control. The default is classDefaultStyle (see page 271).

handle The window handle of an existing entry field control.

Public Members

alignment

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

Returns the current alignment for this entry field object. The returned value is an enumerator provided by Alignment (see page 269).

Motif Information: In AIX, this function always returns the alignment left.

backgroundColor

Condition: (2) OS/2 Only

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

Returns the background color value of the entry field. If you have not set this color, the default is returned.

charType

```
CharType charType() const;
```

Returns the current type of character that this entry field accepts. The returned value is an enumerator provided by CharType (see page 269).

Motif Information: In Motif, an IEntryField is always of type *any*.

clear

```
virtual IEntryField &clear(unsigned long timestamp = 0);
```

Replaces the selected text in the entry field with blanks. If there is no text selected, this function throws an IInvalidRequest exception.

Text can be selected by the user or by your code calling the selectRange function. The code can call the hasSelectedText to determine if any text in the entry field is selected.

To remove the text contents of the entry field, regardless of whether any text is selected, you should call setText, passing it a 0-length string, or removeAll.

copy

```
virtual IEntryField &copy(unsigned long timestamp = 0);
```

Copies the selected text to the clipboard. If there is no text selected, this function throws an IInvalidRequest exception.

Text can be selected by the user or by your code calling the selectRange function. Your code can call the hasSelectedText to determine if any text in the entry field is selected.

To remove the text contents of the entry field, regardless of whether any text is selected, you should call setText, passing it a 0-length string, or removeAll.

cut

```
virtual IEntryField &cut(unsigned long timestamp = 0);
```

Removes the selected text from the entry field control and puts it in the clipboard. If there is no text selected, this function throws an IInvalidRequest exception.

Text can be selected by the user or by your code calling the selectRange function. Your code can call the hasSelectedText to determine if any text in the entry field is selected.

To remove the text contents of the entry field, regardless of whether any text is selected, you should call setText, passing it a 0-length string, or removeAll.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 271) unless you have changed the style using setDefaultStyle (see page 264).

disable
```
virtual IEntryField &disable();
```

Disables an entry field control. When you disable the entry field, the entry field:

- No longer accepts keyboard events
- No longer accepts mouse event
- Is greyed out
- Beeps when the user tries to enter data into the entry field

disableAutoScroll
```
virtual IEntryField &disableAutoScroll();
```

Disables the style autoScroll (see page 271) on an entry field control.

disableAutoTab
```
virtual IEntryField &disableAutoTab();
```

Disables the style autoTab (see page 271) on an entry field control.

disableCommand
```
virtual IEntryField &disableCommand();
```

Disables the style command (see page 271) on an entry field control.

disableDataUpdate
```
virtual IEntryField &disableDataUpdate();
```

Prevents the insertion or changing of characters in the entry field's text.

disableInsertMode
```
virtual IEntryField &disableInsertMode();
```

Enables the entry field for overtype mode and changes the cursor's appearance.

disableMargin
```
virtual IEntryField &disableMargin();
```

Disables the style margin (see page 271) on an entry field control. As a result, the entry field will not have a border surrounding its edit rectangle.

discard
```
virtual IEntryField &discard();
```

Deletes the selected text. If there is no text selected, this function throws an IInvalidRequest exception.

Text can be selected by the user or by your code calling the selectRange function. Your code can call the hasSelectedText to determine if any text in the entry field is selected.

To remove the text contents of the entry field, regardless of whether any text is selected, you should call setText, passing it a 0-length string, or removeAll.

enable

```
virtual IEntryField &enable(Boolean enable = true);
```

Enables or disables an entry field control.

enableAutoScroll

```
virtual IEntryField &enableAutoScroll(
                      Boolean enable = true);
```

Enables or disables the style autoScroll (see page 271) on an entry field control.

enableAutoTab

```
virtual IEntryField &enableAutoTab(Boolean enable = true);
```

Enables or disables the style autoTab (see page 271) on an entry field control.

enableCommand

```
virtual IEntryField &enableCommand(Boolean enable = true);
```

Enables or disables the style command (see page 271) on an entry field control.

enableDataUpdate

```
virtual IEntryField &enableDataUpdate(
                      Boolean update = true);
```

Enables or disables the read-only mode on an entry field control.

enableInsertMode

```
virtual IEntryField &enableInsertMode(
                      Boolean insert = true);
```

Enables the entry field for insert mode and changes the cursor's appearance.

enableMargin

```
virtual IEntryField &enableMargin(Boolean enable = true);
```

Enables or disables the style margin (see page 271) on an entry field control. As a result, this function adds or removes a border from the edit area.

enableNotification

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

Causes the entry field control to send notifications to any observer objects added.

foregroundColor

Condition: (2) OS/2 Only

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

Returns the foreground color value of the entry field.

hasChanged

```
Boolean hasChanged() const;
```

Queries whether the entry field text has changed since the last query.

hasSelectedText

```
Boolean hasSelectedText() const;
```

Queries whether any of the entry field text is selected.

isAutoScroll

```
Boolean isAutoScroll() const;
```

Queries whether the style autoScroll (see page 271) is set on an entry field control.

isAutoTab

```
Boolean isAutoTab() const;
```

Queries whether the style autoTab (see page 271) is set on an entry field control.

isCommand

```
Boolean isCommand() const;
```

Queries whether the style command (see page 271) is set on an entry field control.

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

Queries whether the entry field is empty.

isInsertMode
```
Boolean isInsertMode() const;
```

Queries whether the entry field is in insert mode.

isMargin
```
Boolean isMargin() const;
```

Returns a Boolean depending on whether the control margin (see page 271) has a surrounding border.

isReadOnly
```
Boolean isReadOnly() const;
```

Queries whether the entry field control is in read-only mode.

leftIndex
```
unsigned long leftIndex() const;
```

Returns the index of the first character displayed at the left edge of the entry field control. The index is 0-based.

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

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.

Presentation Manager Information: The default text limit on OS/2 is 32.

moveSizeTo
Condition: (3) AIX only

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

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.

paste

```
virtual IEntryField &paste();
```

Copies text from the clipboard to the entry field control, replacing any selected text. If the clipboard does not contain any text, this function throws an IInvalidRequest exception.

Text can be selected by the user or by your code calling the selectRange function. Your code can call the hasSelectedText to determine if any text in the entry field is selected.

To remove the text contents of the entry field, regardless of whether any text is selected, you should call setText, passing it a 0-length string, or removeAll.

removeAll

```
virtual IEntryField &removeAll();
```

Deletes the entire contents of the entry field control.

selectedRange

```
IRange selectedRange() const;
```

Returns the range of the selected text. If no text is selected, this function returns a range of (cursor position, cursor position). The selected range is the 0-based index of the first character selected and the 0-based index of the last character selected.

selectedText

```
IString selectedText() const;
```

Returns the selected text string.

selectRange

```
virtual IEntryField &selectRange(
                      const IRange &range = IRange(0,end),
                      unsigned long timestamp = 0);
```

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.

setAlignment

Condition: (1) Ignored on AIX

```
virtual IEntryField &setAlignment(Alignment alignment);
```

Sets the alignment of the entry field control.

alignment Use the enumeration Alignment (see page 269) to specify the alignment of the entry field.

Motif Information: In AIX, you can only set the Alignment to left.

setChangedFlag

```
virtual IEntryField &setChangedFlag(
                      Boolean changeFlag = true);
```

Sets the entry field so that the control considers the text to have changed (or not) since the last query.

setCharType

Condition: (1) Ignored on AIX

```
virtual IEntryField &setCharType(CharType type);
```

Sets the type of character the entry field control can accept.

type Use the enumeration CharType (see page 269) to specify the type of characters in the entry field.

Motif Information: AIX only supports the CharType enumerator any. This type corresponds to X multi-byte character set (MBCS).

setDefaultStyle

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

Sets the default style for all subsequent entry fields.

style Use the styles provided by IEntryField::Styles (see page 271) to specify the default style.

setLeftIndex

```
virtual IEntryField &setLeftIndex(unsigned long index);
```

Sets the first displayed character at the left edge of the entry field control. The index is 0-based.

setLimit

```
virtual IEntryField &setLimit(unsigned long textLimit);
```

Condition: (2) OS/2 Only

```
virtual IEntryField &setLimit(const IResourceId &textLimit);
```

Sets the maximum number of bytes the entry field can contain. If the entry field already contains more characters than the proposed limit, this function throws an IInvalidParameter exception.

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.

text

```
virtual IString text() const;
```

Returns the text of the entry field's control window.

Figure 84. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl

Figure 84. Inherited Public Members

Member function	Class
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow

Figure 84. Inherited Public Members

Member function	Class
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow

Figure 84. Inherited Public Members

Member function	Class
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size that this entry field control can be based on the text string length limit and the current font.

initialize

Condition: (3) AIX only

```
IEntryField &initialize(unsigned long style);
```

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 (see page 951) or IWindow::create (see page 946) has been called.

passEventToOwner

Condition: (3) AIX only

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

Determines whether an event is passed to the owner of this control.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

setLayoutDistorted

Condition: (3) AIX only

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

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

setStyle

Condition: (2) OS/2 Only

```
virtual IEntryField &setStyle(unsigned long style);
```

Replaces the style of an entry field control, while preserving the cursor position and selected text.

unregisterCallbacks

Condition: (3) AIX only

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

Removes any callbacks and X event handlers added by the registerCallbacks function.

Figure 85. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow

Figure 85. Inherited Protected Members

Member function	Class
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

```
public:
enum 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 Information: X/Motif only supports the left alignment.

CharType

```
public:
enum CharType
  {
  sbcs,
  dbcs,
  any,
  mixed = oem,
  };
```

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.

mixed
Accepts text that is a mixture of SBCS and double-byte character set (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 data because of the addition of shift-in/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 Information: X/Motif only support the enumerator any.

Chapter 61. IEntryField::Style

Derivation

IBase
  IBitFlag
    IEntryField::Style

Header File: ientryfd.hpp

The nested class IEntryField::Style provides a set of valid entry field styles for the static member functions of the class IEntryField (see page 256). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

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.

Note: If the text contains both SBCS and DBCS characters and is converted from an ASCII code page into an EBCDIC code page, this style causes an entry field to not account for shift-in and shift-out characters that are introduced into its text.

autoScroll
Specifies that 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.

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.

centerAlign
Centers the text in the entry field.

classDefaultStyle
Specifies the original default style for this class, which is leftAlign | anyData | margin | autoScroll | IWindow::visible.

command
Creates information used by the Information Presentation Facility (IPF) to provide command help if the user requests help for this field.

Note: You must not give this style to more than one entry field on each frame.

© Copyright IBM Corp. 1994

dbcsData
Sets the entry field to accept double-byte text only.

leftAlign
Left-justifies the text in the entry field.

margin
Draws a border around the entry field and puts a margin between the border and the entry field. The font currently used for entry field text determines the margin's size. Typically, the margin is half a character-width wide and half a character-height high.

Because the border and its margin is drawn around the entry field, this style does not change the position of the entry field itself. For example, if an entry field is positioned at (100,100) and has a margin and border with height and width of 2, the lower-left corner of the border is positioned at (98,98).

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 an increase in the length of the data because of the addition of shift-in or shift-out characters, but does not exceed the text limit of the entry field.

readOnly
Prevents insertions or changes to the text in the control.

rightAlign
Right-justifies the text in the entry field.

sbcsData
Sets the entry field to accept single-byte text only.

unreadable
Displays each character of the text as an asterisk.

Motif Information: 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.

Public Members

Figure 86. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag

Figure 86. Inherited Public Members

Member function	Class
operator !=	IBitFlag
operator <<	IBase*

Figure 86. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 87. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 62. IEnumHandle

Derivation

IBase
  IHandle
    IEnumHandle

Header File: ihandle.hpp

Objects of the IEnumHandle class access lists of enumerated items. For example, an enumeration list can contain the handles of all the child windows for a given parent window.

Motif Information: AIX does not support this class.

Presentation Manager Information: IEnumHandle is an alias for the OS/2 Programmer's Toolkit type HENUM.

Constructors

```
public:
IEnumHandle(Value henum = 0);
```

You can construct objects of this class from an enumeration handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IEnumHandle value.

Figure 88. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 63. IEvent

Derivation

IBase
  IVBase
    IEvent

Header File: ievent.hpp

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.

Portability Considerations: The User Interface Class Library-supported event types (IEvent and derived classes) provide portable accessors to obtain event details, and 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 use the consistent definition of the base IEvent structure. The library also provides non-portable 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.

Motif Information: This section describes how X/Motif events and their data are used to construct IEvent objects. This information is primarily for those who wish to perform X/Motif unique extensions or integrate custom widgets.

The basic Motif callback function signature is:

```
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. XmAnyCallbackStruct can be used to refer to the above structure.

An Xt event registration callback function signature is:

```
void  eventHandlerFn(
         Widget    widget,  /* widget in whose window event occurred*/
         caddr_t   client_data,  /* client data                    */
         XEvent   *event,        /* event data                     */
         ::Boolean continue_to_dispatch);
```

© Copyright IBM Corp. 1994

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.

Constructors

```
public:
IEvent(
   const IWindowHandle &handle,
   unsigned long eventId,
   const IEventParameter1 &parameter1,
   const IEventParameter2 &parameter2);

public:
IEvent(
   IWindow *window,
   unsigned long eventId,
   const IEventParameter1 &parameter1,
   const IEventParameter2 &parameter2);

public:
IEvent(const IEvent &event);
```

You can construct objects of this class in the following ways:

- From an IHandle (see page 400) object, an event ID, an IEventParameter1 object, and an IEventParameter2 object.
- From an object of IWindow (see page 924), an event ID, an IEventParameter1 object, and an IEventParameter2 object.
- From another IEvent.

In AIX, the IEvent constructor interprets its parameters as follows:

handle
: The widget parameter, as passed to the callback or event handler function.

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.

Public Members

controlHandle
```
IWindowHandle controlHandle() const;
```

Returns the IWindowHandle (see page 960) object for the presentation window which 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 (see page 71) to an IHandler (see page 402), the return value is the same as that for dispatchingWindow. If the event has been owner-routed and dispatched from the owner of the ICheckBox, the return value is different. controlWindow 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

Motif Information: On AIX, this function returns valid value for all event types.

controlWindow
```
virtual IWindow *controlWindow() const;
```

Returns a pointer to the IWindow (see page 924) object which 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 (see page 71) to an IHandler (see page 402), the return value is the same as that for dispatchingWindow. If the event has been owner-routed and dispatched from the owner of the ICheckBox, the return value is different. controlWindow 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

Motif Information: On AIX, this function returns valid value for all event types.

dispatchingWindow

```
virtual IWindow *dispatchingWindow() const;
```

Returns a pointer to the IWindow (see page 924) object from which this event was just dispatched. The pointer can represent any of the following:

- The original object which 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.

eventId

```
unsigned long eventId() const;
```

Returns the ID for the event.

Motif Information: 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 a Xt event registration routine.

eventType

Condition: (3) AIX only

```
EventType eventType() const;
```

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 library. The event type is determined and set by IWindow (see page 924). You should determine the event type before checking the IEvent::eventId (see page 278) because the identifiers may overlap for different event types.

handle

```
IWindowHandle handle() const;
```

Returns an IWindowHandle (see page 960) object for the IWindow (see page 924) object the event has been dispatched to. Use of this function may result in non-portable code; use controlHandle or dispatchingHandle instead.

parameter1

```
IEventParameter1 parameter1() const;
```

Returns the first event parameter.

Motif Information: The return value is either:

- A pointer to the Motif callback structure
- A NULL, indicating an XEvent.

parameter2

```
IEventParameter2 parameter2() const;
```

Returns the second event parameter.

Motif Information: 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

Condition: (3) AIX only

```
Boolean passToOwner() const;
```

IWindow::dispatch (see page 947) calls this member function to determine if an IEvent object can be passed up the owner chain.

result

```
IEventResult result() const;
```

Returns the event result.

setControlHandle

Condition: (4) Typically for internal use

```
IEvent &setControlHandle(const IWindowHandle &handle);
```

Sets the IWindowHandle (see page 960) object for the presentation window which 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 (see page 71) to an IHandler (see page 402), this value should be the same as that for dispatchingWindow. 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.

setDispatchingHandle

Condition: (4) Typically for internal use

```
IEvent &setDispatchingHandle(const IWindowHandle &handle);
```

Sets the IWindowHandle (see page 960) object for the IWindow (see page 924) 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 which 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.

setEventType

Condition: (3) AIX only

```
IEvent &setEventType(EventType typeOfEvent);
```

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.

setHandle

Condition: (4) Typically for internal use

```
IEvent &setHandle(const IWindowHandle &handle);
```

Sets the IWindowHandle (see page 960) object for the IWindow (see page 924) 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 non-portable code; use setControlHandle or setDispatchingHandle instead.

setPassToOwner

Condition: (3) AIX only

```
IEvent &setPassToOwner(Boolean passOn = true);
```

Controls whether or not the IEvent object can propagate up the owner chain.

setResult

```
IEvent &setResult(Boolean eventResult);

IEvent &setResult(const IEventResult &eventResult);
```

Sets the event result. The IEventResult type is defined in the header file ievtdata.hpp and is part of the class IEventData (see page 282).

window

```
virtual IWindow *window() const;
```

Returns a pointer to the IWindow (see page 924) object the event has been dispatched to. Use of this function may result in non-portable code; use controlWindow or dispatchingWindow instead.

Figure 89. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

EventType

Condition: (3) AIX only

```
public:
enum EventType
  {
  Xevent,
  MotifCallback,
  Library,
  };
```

The preceding enumerators specify the origin of the event that triggered the creation of the IEvent object.

Chapter 64. IEventData

Derivation

IBase
  IEventData

Header File: ievtdata.hpp

Use the IEventData class to encapsulate the data of an event. IEventData is interchangable with the following classes, which are actually aliases (that is, typedefs) for IEventData:

- IEventParameter1
- IEventParameter2
- IEventResult
- IHighEventParameter
- ILowEventParameter

Portability Considerations: The interpretation of the IEventData objects in an IEvent are system dependent. See IEvent (see page 275) 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.

Motif Information: 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.

Constructors

```
public:
IEventData();

public:
IEventData(void *value);

public:
IEventData(unsigned long value);

public:
IEventData(int value);

public:
IEventData(BooleanConstants value);

public:
IEventData(unsigned short lowValue, unsigned short hiValue);
```


© Copyright IBM Corp. 1994

```
public:
IEventData(
   unsigned short lowValue,
   char lowByte,
   char hiByte);
```

You can construct objects of this class in the following ways:

- Using the default constructor. The event data is set to 0.
- From a pointer to void. The event data is set to the specified void pointer.
- From an unsigned long integer value. The event data is set to the specified unsigned long integer.
- From an integer. The event data is set to the specified integer. This constructor accepts an uncasted value of 0.
- 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.
- From one unsigned short integer 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.

Public Members

asLong
```
long asLong() const;
```

Returns the long event data value.

asUnsignedLong
```
unsigned long asUnsignedLong() const;
```

Returns the unsigned long event data value.

char1
```
char char1() const;
```

Returns the character in the low byte of the event data's low word. This is the same as lowLowByte (see page 284).

char2
```
char char2() const;
```

Returns the character in the high byte of the event data's low word. This is the same as lowHighByte (see page 284).

char3
```
char char3() const;
```

Returns the character in the low byte of the event data's high word. This is the same as highLowByte (see page 284).

char4
```
char char4() const;
```

Returns the character in the high byte of the event data's high word. This is the same as highHighByte (see page 284).

highHighByte
```
char highHighByte() const;
```

Returns the character in the high byte of the event data's high word. This is the same as char4 (see page 284).

highLowByte
```
char highLowByte() const;
```

Returns the character in the low byte of the event data's high word. This is the same as char3 (see page 283).

highNumber
```
unsigned short highNumber() const;
```

Returns the event data's high word. This is the same as number2 (see page 285).

lowHighByte
```
char lowHighByte() const;
```

Returns the character in the high byte of the event data's low word. This is the same as char2 (see page 283).

lowLowByte
```
char lowLowByte() const;
```

Returns the character in the low byte of the event data's low word. This is the same as char1 (see page 283).

lowNumber
```
unsigned short lowNumber() const;
```

Returns the event data's low word. This is the same as number1 (see page 285).

number1

```
unsigned short number1() const;
```

Returns the event data's low word. This is the same as lowNumber (see page 284).

number2

```
unsigned short number2() const;
```

Returns the event data's high word. This is the same as highNumber (see page 284).

operator char *

```
operator char *() const;
```

Returns the event data as a pointer to a character.

operator unsigned long

```
operator unsigned long() const;
```

Returns the event data as an unsigned long number.

Figure 90. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 65. IFileDialog

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow
          IFileDialog
```

Header File: ifiledlg.hpp

Objects of the IFileDialog class display a file dialog for the user to choose a file. Once the user has chosen a file, you can use member functions of this class to retrieve information about the chosen file. You can use the following handlers to process the appropriate events for this control:

- ICommandHandler (see page 131)
- IFileDialogHandler (see page 303)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)

Nested Classes

IFileDialog::Settings
IFileDialog::Style

Constructors

```
public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   const Style &style = defaultStyle ( ));

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   const Settings &settings,
   const Style &style = defaultStyle ( ));

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   const Style &style,
   const Settings &settings);
```


© Copyright IBM Corp. 1994

```
public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   IHandler *handler,
   const Style &style = defaultStyle ( ),
   const Settings &settings = Settings ( ));

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   Style style = defaultStyle ( ));

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   const Settings &settings,
   Style style = defaultStyle ( ));

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   Style style,
   const Settings &settings);

public:
IFileDialog(
   IWindow *parent,
   IWindow *owner,
   IHandler *handler,
   Style style = defaultStyle ( ),
   const Settings &settings = Settings ( ));
```

You can construct objects of this class in the following ways:

- Using constructors that accept a parent, owner, style, and settings as parameters

  Note: In OS/2 Presentation Manager, you can have modal windows that do not use the desktop as their parent. However, Presentation Manager can change the owner window of your dialog to something other than what you specified to prevent your application from being hung. See the *Presentation Manager Programming Reference Volume 2* for more information under WinLoadDlg.

  The easiest way to ensure that your dialog is modal is to use the desktop as the *parent* and an application window as the *owner*. Presentation Manager only disables the owner window and its child windows (the child windows of the child windows, and so forth) while a modal dialog is displayed. If you specify 0 for *parent*, the desktop automatically becomes the parent of the dialog.

- Adding an IFileDialogHandler (see page 303) to a modal IFileDialog. You must specify the handler on the constructor. Once you create a modal IFileDialog, the dialog does not return control to the application until the user closes the dialog. At that time, it is too late to add a handler.

The parameters for these constructors are the following:

parent The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.

owner The owner window.

style (Optional) Use the styles provided by IFileDialog::Styles (see page 299) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style.

settings (Optional) Use the settings provided by IFileDialog::Settings (see page 294) to specify the setting.

handler A pointer to an IHandler (see page 402) object.

Public Members

buttonPressedId
```
unsigned long buttonPressedId() const;
```

Returns the ID of the push button used to close the dialog.

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

Returns the default style. The default style is classDefaultStyle (see page 299) unless you changed the style using setDefaultStyle (see page 289).

fileName
```
IString fileName(unsigned fileNumber = 0) const;
```

Returns the fully qualified file name selected by the user.

fileNumber The number indicating which file name you want. The default is 0. When you specify the style multiSelection (see page 299) for your IFileDialog object, set *fileNumber* to selectedFileCount (see page 289) minus 1 to ensure the correct number of files is returned.

isModeless
```
Boolean isModeless() const;
```

If this is a modeless dialog, true is returned.

pressedOK
```
Boolean pressedOK() const;
```

If the user ends the dialog by selecting the **OK** push button, true is returned.

returnValue

```
long returnValue() const;
```

If an error occurs, the return code is returned.

Motif Information: This function always returns 0.

Presentation Manager Information: The return value is an FDS_ERR system return code. When a dialog fails, this value tells the application the reason for the failure.

saveAsEAType

```
IString saveAsEAType() const;
```

Returns the extended attribute (EA) for a Save As dialog. This is the type the user chose from the drop-down list box of EA types. If the user chose the EA type <All Files>, a 0-length IString is returned.

Motif Information: This function always returns an empty string.

selectedFileCount

```
unsigned selectedFileCount() const;
```

Returns the number of files selected by the user.

Note: If you create the dialog with the style multiSelection (see page 299), this number can be more than 1.

setDefaultStyle

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

Sets the default style for all subsequent file dialogs.

style Dialog style. Use the styles provided by IFileDialog::Styles (see page 299) to specify the default style.

Figure 91. Inherited Public Members

Member function	Class
activeColor	IWindow
addExtension	IFrameWindow
addToWindowList	IFrameWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow

Figure 91. Inherited Public Members

Member function	Class
backgroundColor	IFrameWindow
beginFlashing	IFrameWindow
borderColor	IWindow
borderHeight	IFrameWindow
borderSize	IFrameWindow
borderWidth	IFrameWindow
characterSize	IWindow
childAt	IWindow
client	IFrameWindow
clientHandle	IFrameWindow

Figure 91. Inherited Public Members

Member function	Class
clientRectFor	IFrameWindow
close	IFrameWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IFrameWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
dismiss	IFrameWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IFrameWindow
enableUpdate	IWindow
endFlashing	IFrameWindow
font	IWindow
foregroundColor	IWindow
frameRectFor	IFrameWindow
handle	IWindow
handleException	IWindow
handleFor	IFrameWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
icon	IFrameWindow
id	IWindow
inactiveColor	IWindow
isAnExtension	IFrameWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFlashing	IFrameWindow
isFrameWindow	IWindow

Figure 91. Inherited Public Members

Member function	Class
isGroup	IWindow
isLayoutDistorted	IWindow
isMaximized	IFrameWindow
isMinimized	IFrameWindow
isModal	IFrameWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
maximize	IFrameWindow
maximizeRect	IFrameWindow
messageQueue	IWindow
minimize	IFrameWindow
minimizeRect	IFrameWindow
minimumSize	IWindow
mousePointer	IFrameWindow
moveSizeTo	IWindow
moveSizeToClient	IFrameWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
notifyOwner	IFrameWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeExtension	IFrameWindow
removeFromWindowList	IFrameWindow

Figure 91. Inherited Public Members

Member function	Class
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
restore	IFrameWindow
restoreRect	IFrameWindow
result	IFrameWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setBorderHeight	IFrameWindow
setBorderSize	IFrameWindow
setBorderWidth	IFrameWindow
setClient	IFrameWindow
setDestroyOnClose	IFrameWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setExtensionSize	IFrameWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setIcon	IFrameWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	IFrameWindow
setMinimumSize	IWindow

Figure 91. Inherited Public Members

Member function	Class
setMousePointer	IFrameWindow
setOwner	IWindow
setParent	IWindow
setRestoreRect	IFrameWindow
setResult	IFrameWindow
setShadowColor	IWindow
setToolBarList	IFrameWindow
shadowColor	IWindow
shareParentDBCSStatus	IFrameWindow
show	IWindow
show	IFrameWindow
showModally	IFrameWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
toolBarList	IFrameWindow
topHandle	IWindow
topHandle	IFrameWindow
update	IFrameWindow
useExtensionMinimumSize	IFrameWindow
usesDialogBackground	IFrameWindow
visibleRectangle	IWindow
willDestroyOnClose	IFrameWindow

Note: * Application Support Class Library

Protected Members

registerCallbacks

Condition: (3) AIX only

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

Adds callbacks and X event handlers to this IFileDialog for events it might receive. This function adds all possible callbacks and X event handlers. IHandler derived classes later determine which events they process.

unregisterCallbacks

Condition: (3) AIX only

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

Removes callbacks and X event handlers from this IFileDialog which were added by registerCallbacks (see page 292).

Figure 92. Inherited Protected Members

Member function	Class
addDefaultHandler	IFrameWindow
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
attachClient	IFrameWindow
calcMinimumSize	IWindow
create	IWindow
create	IFrameWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
extensions	IFrameWindow
findExtension	IFrameWindow
findExtensions	IFrameWindow
initialize	IFrameWindow
isFrameWindow	IFrameWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
isRelatedHandle	IFrameWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier

Figure 92. Inherited Protected Members

Member function	Class
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IFrameWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeDefaultHandler	IFrameWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setExtensions	IFrameWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
start	IFrameWindow
startHandlingEventsFor	IWindow
style	IWindow
tryToLoadDialog	IFrameWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IFrameWindow
windowSystemClass	IFrameWindow

Figure 92. Inherited Protected Members

Member function	Class
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 66. IFileDialog::Settings

Derivation

IBase
  IFileDialog::Settings

Header File: ifiledlg.hpp

Objects of the IFileDialog::Settings class contain input for an IFileDialog (see page 286) object. If you do not specify a Settings object on the IFileDialog constructor, a default Settings object is created for you. This default object provides an open dialog with no other settings in effect.

You can pass the following information within an IFileDialog::Settings object to an IFileDialog:

- The dialog's title
- The text to appear on the **OK** push button
- The position of the dialog
- Whether the dialog is an Open or Save As dialog
- An initial file specification

Presentation Manager Information: You can also specify the following items within a Settings object:

- An initial file type extended attribute
- An initial drive specification
- A dialog template to use in place of the system default

Constructors

```
public:
Settings();
```

You can only construct objects of this class by using the default constructor, which accepts no parameters.

Public Members

addDrive

Condition: (1) Ignored on AIX

```
Settings &addDrive(const char *drive);
```


© Copyright IBM Corp. 1994

Adds a drive or network identifier to the Drive list in the initial dialog. If you call this function, the Drive list is limited to the drives you specify in *drive*.

drive The drive or network identifier to add to the Drive list.

addFileType

Condition: (1) Ignored on AIX

```
Settings &addFileType(const char *fileType);
```

Adds the specified type to the drop-down list box of extended-attribute types.

fileType The extended-attribute type to add.

fileName

```
IString fileName() const;
```

Returns the default file name selection.

initialDrive

```
IString initialDrive() const;
```

Returns the drive for which initial information is displayed.

initialFileType

```
IString initialFileType() const;
```

Returns the file type that is used to filter the initial list of files.

isDialogTemplateSet

```
Boolean isDialogTemplateSet() const;
```

Returns true if the dialog template was set.

isOpenDialog

```
Boolean isOpenDialog() const;
```

Returns true if dialog is an open dialog.

isPositionSet

```
Boolean isPositionSet() const;
```

Returns true if the dialog position was set.

OKButtonText

```
IString OKButtonText() const;
```

Returns the text of the **OK** button.

position

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

Returns the initial placement of the dialog within the parent window.

setDialogTemplate

Condition: (2) OS/2 Only

```
Settings &setDialogTemplate(const IResourceId &resId);
```

Sets a dialog template resource to use in place of the OS/2-supplied default font dialog.

resId The resource ID of the dialog template to use.

setFileName

```
Settings &setFileName(const char *fileName);
```

Sets a default file name selection.

fileName The file name to use as the default. If you want to specify wildcard characters or path information for the dialog defaults, include this information in this parameter's string. If you do not specify the path with the file name, the current directory is used.

setInitialDrive

Condition: (1) Ignored on AIX

```
Settings &setInitialDrive(const char *drive);
```

Sets the drive for which initial information is displayed. If you do not call this function, the dialog displays the information for the current drive. If you specify a drive for this function, you do not have to specify it again when calling addDrive (see page 294).

drive The initial drive to display.

setInitialFileType

Condition: (1) Ignored on AIX

```
Settings &setInitialFileType(const char *fileType);
```

Sets the extended attribute that filters the initial display of files. If you do not call this function, the default is <All Files>. If you specify a type for this function, you do not have to specify it again when calling addFileType (see page 295).

fileType The initial extended attribute.

setOKButtonText

```
Settings &setOKButtonText(const char *newText);

Settings &setOKButtonText(const IResourceId &resId);
```

Sets the text that appears on the **OK** push button.

newText A character string to appear on the **OK** push button.

resId The resource ID of a string to appear on the **OK** push button.

setOpenDialog

```
Settings &setOpenDialog();
```

Creates an Open File dialog. An Open File dialog has the list of files enabled for selection, while a Save As dialog has this list disabled. An Open File dialog is the default.

setPosition

```
Settings &setPosition(const IPoint &position);
```

Sets the initial placement of the dialog within the parent window.

position The initial position for the dialog.

setSaveAsDialog

```
Settings &setSaveAsDialog();
```

Creates a **Save As** dialog. An **Open File** dialog has the list of files enabled for selection, while a **Save As** dialog has this list disabled. An **Open File** dialog is the default. You can use the style IFileDialog::selectableListBox (see page 299) to force the list of files to be selectable on a **Save As** dialog.

setTitle

```
Settings &setTitle(const char *newTitle);

Settings &setTitle(const IResourceId &resId);
```

Sets the dialog's title. The parameters are the following:

newTitle A character string to use as the dialog title.

resId The resource ID of a string to use as the dialog title.

title

```
IString title() const;
```

Returns the dialog's title.

Figure 93. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 67. IFileDialog::Style

Derivation

IBase
  IBitFlag
    IFileDialog::Style

Header File: ifiledlg.hpp

The nested class IFileDialog::Style provides a set of valid file dialog styles for the member functions of the class IFileDialog (see page 286).

Styles

applyButton
Adds an **Apply** push button to the dialog. This is useful in a modeless dialog.

classDefaultStyle
Specifies the original default style for this class, which is noStyle. This style results in a modal open dialog.

filter
Specifies that the dialog use the union of the string filter and the extended-attribute type filter when filtering files for the Files list box. When you do not specify this style, the list box uses the intersection of the two by default.

helpButton
Adds a **Help** push button to the dialog. This push button sends a help message to the owner of the dialog.

includeEAS
Specifies that the dialog always queries the extended attribute information for files as it fills the Files list box. The default is to not query the information, unless the user selects an extended-attribute type filter from the dialog.

modeless
Specifies that the dialog is modeless. The constructor for a modeless dialog returns immediately. To determine which files were chosen by the user, create an IFileDialogHandler and use IFileDialogHandler::modelessResults (see page 305).

multiSelection
Specifies that the Files list box for the dialog is a multiple-selection list box. If you do not specify this style, the default is a single-selection list box.

noStyle
Disables all of the valid styles. Use this style *only* when you are not using any other styles.

© Copyright IBM Corp. 1994

preload
Specifies that the dialog do the following:

- Preload the volume information for the drives
- Preset the current default directory for each drive

By default, the volume label is blank and the initial directory is the root directory for each drive.

selectableListbox
Enables the Files list box on a Save As dialog. If you do not specify this style, it disables the Files list box for a Save As dialog. Disabling is the default.

Motif Information: For the AIX release, applyButton does not add the **Apply** push button to the dialog. Instead, this style specifies that the dialog not be dismissed when the user presses the **OK** push button.

The AIX release ignores the following styles:

- filter
- includeEAS
- preload

Public Members

Figure 94. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 95. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 68. IFileDialogEvent

Derivation

IBase
  IVBase
    IEvent
      IFileDialogEvent

Header File: ifilehdr.hpp

Objects of the IFileDialogEvent class encapsulate events used in an IFileDialogHandler (see page 303) object. It provides the name and length of the file that the event pertains to.

Constructors

```
public:
IFileDialogEvent(IEvent &event);
```

Although you can construct objects of this class, typically IFileDialogHandler::dispatchHandlerEvent (see page 304) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

fileLength

```
unsigned long fileLength() const;
```

Returns the length of the file name.

fileName

```
IString fileName() const;
```

Returns the file name the event pertains to.

Figure 96. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent

Figure 96. Inherited Public Members

Member function	Class
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*

© Copyright IBM Corp. 1994

Figure 96. Inherited Public Members

Member function	Class
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 69. IFileDialogHandler

Derivation

IBase
  IVBase
    IHandler
      IFileDialogHandler

Header File: ifilehdr.hpp

Objects of the IFileDialogHandler class handle the various events that affect the file dialog.

You can create a handler derived from IFileDialogHandler and attach it to a file dialog. Do this by calling IHandler::handleEventsFor (see page 403) to pass the appropriate file dialog to the file dialog handler.

Note: To add a handler to a modal file dialog, you must specify a pointer to the handler using the IFileDialog constructor (see page 286) that accepts such a pointer as one of its parameters.

When the file dialog handler receives a file dialog event, it creates an IFileDialogEvent (see page 301) object and routes that object to the appropriate IFileDialogHandler virtual function. You can override these virtual functions to supply your own specialized processing of a file dialog event.

The return value from the virtual functions specifies whether the file dialog event is passed on for additional processing, as follows:

true The file dialog event requires no additional processing. Do not pass it to another handler.

false Pass the file dialog event to the next handler for additional processing, as follows:

- If there is another handler for the file dialog, pass the file dialog event to the next handler.
- If this is the last handler for the file dialog, call IWindow::defaultProcedure (see page 946) to process the file dialog event.

Constructors

```
public:
IFileDialogHandler();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

© Copyright IBM Corp. 1994

Public Members

handleEventsFor

```
virtual IFileDialogHandler &handleEventsFor(
                              IFileDialog *fileDialog);
```

Attaches the handler to the specified file dialog passed in the argument.

stopHandlingEventsFor

```
virtual IFileDialogHandler &stopHandlingEventsFor(
                              IFileDialog *fileDialog);
```

Detaches the handler from the specified file dialog passed in the argument.

Figure 97. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a file dialog event is received, the appropriate virtual function is called.

filter

```
virtual Boolean filter(IFileDialogEvent &event);
```

Lets you determine which file names appear in the list box. Use IFileDialogEvent::fileName (see page 301) to determine the file that is about to be added to the dialog's file list box. If you do not want the file to appear in the list box,

call IEvent::setResult(false) (see page 281). If you do not want the event to be processed by any other handlers, return true.

filterName

```
virtual Boolean filterName(
                  const IString &fileName,
                  IFileDialog *dialog);
```

Override this function to supply an easier way of filtering files. The supplied default function returns true. If you do not override it, nothing is filtered. If you do not want *fileName* to appear in the dialog, return false.

fileName The name of the file you want to filter.

dialog File dialog pointer. If you need to access the IFileDialog object itself, you can use the IFileDialog* (see page 286) pointer.

modelessResults

```
virtual Boolean modelessResults(IFileDialog *endingDialog);
```

Called when a modeless dialog is closed, even if the dialog is cancelled. This is done so that you can destroy the IFileDialog (see page 286), if needed. Use IFileDialog::pressedOK (see page 288) to determine whether the user ended the dialog by pressing the **OK** push button.

If you need to destroy the IFileDialog object, do not delete the IFileDialog* pointer from within modelessResults. Instead, use the following:

```
endingDialog->setAutoDeleteObject(True)
```

validate

```
virtual Boolean validate(IFileDialogEvent &event);
```

Called when a user selects a file and presses **Enter** or double-clicks on a file in the list box. Use IFileDialogEvent::fileName (see page 301) to get the name selected by the user. If the name is unacceptable, call IEvent::setResult(false) (see page 281). In this case, the dialog is not ended. If you do not want the event to be processed by any other handlers, return true.

validateName

```
virtual Boolean validateName(
                  const IString &fileName,
                  IFileDialog *dialog);
```

Override this function to supply an easier way of validating. The supplied default function returns true, so that if you do not override it, nothing is validated. If *fileName* is unacceptable, return false.

fileName The name of the file you want to validate.

dialog If you want to inform the user why you rejected the name, use the IFileDialog* pointer in an IMessageBox (see page 525) constructor.

Figure 98. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 70. IFocusHandler

Derivation

IBase
  IVBase
    IHandler
      IFocusHandler

Header File: ifocshdr.hpp

Objects of the IFocusHandler class handle events occuring 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 (see page 149)
IEntryField (see page 256)
IListBox (see page 458)
IMultiLineEdit (see page 557)
ISlider (see page 792)
ISpinButton (see page 803)
IBaseSpinButton (see page 32)
INumericSpinButton (see page 612)
ITextSpinButton (see page 878)

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 (see page 403) 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 (see page 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:

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 (see page 947) to dispatch the focus event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (see page 946) to process the focus event.

© Copyright IBM Corp. 1994

Motif Information: In Motif, IFocusHandlers cannot handle the processing of focus change events for ISlider (see page 792) controls.

Constructors

```
public:
IFocusHandler();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

Public Members

Figure 99. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a focus event is received, the appropriate virtual function is called.

gotFocus

```
virtual Boolean gotFocus(IControlEvent &event);
```

Implemented by a derived class to process a focus gain event.

lostFocus

```
virtual Boolean lostFocus(IControlEvent &event);
```

Implemented by a derived class to process a focus loss event.

Note: It is possible that a set focus event can be ahead of a related lost focus event on the message queue.

Figure 100. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 71. IFont

Derivation

IBase
  IVBase
    IFont

Header File: ifont.hpp

Objects of the IFont class manage the use of fonts. Use these objects to select a font through the IFont functions. You can also use the font dialog to:

- Get font information
- Set the font when drawing text

IFont attempts to match the requested font. If it cannot find an exact match, IFont uses the nearest match.

Note: An IFont object represents a particular font that is available on the system. It does not represent:

- A font actually being used by a control
- The currently selected font for a presentation space

IWindow::setFont (see page 940) changes the font used to draw text in a control.

beginUsingFont (see page 312) causes all subsequent text drawn to the specified presentation space to be in the font represented by the IFont object. You can use beginUsingFont in conjunction with an IPaintHandler (see page 647) object.

Portability Considerations: If you are porting an OS/2 Presentation Manager program to Motif, be aware that the font names vary between these systems. The only fonts that you can be reasonably sure of finding on both Motif and Presentation Manager are Courier and Helvetica. On either system, if you try to construct an IFont object for a nonexistent font, you will get Courier as the default font.

Motif Information: Some of the “set” and “use” functions, such as setCharHeight and useBitmapOnly, accept an optional parameter of type IPresSpaceHandle (see page 664). Because the AIX version of IFont does *not* support printer fonts, there is no need to supply the optional IPresSpaceHandle parameter.

The functions in this class that refer to vector fonts use scalable fonts in the X/Motif environment.

Presentation Manager Information: Some of the “set” and “use” functions (such as setCharHeight and useBitmapOnly) accept an optional parameter of type IPresSpaceHandle (see page 664). If you are are using IFont to represent printer fonts,

© Copyright IBM Corp. 1994

you should provide this parameter. If you do not specify a presentation space, IFont uses the presentation space of the desktop.

Constructors

```
public:
IFont(const IWindow *window = 0);

public:
IFont(
   const char *faceName,
   unsigned long pointSize = 0,
   Boolean useFixedFont = false,
   Boolean useVectorFont = false,
   const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));

public:
IFont(const IPresSpaceHandle &presSpaceHandle);

public:
IFont(const IFont &fntCopy);

public:
IFont(
   const char *faceName,
   unsigned long ulPointSize = 0,
   Boolean useFixedFont = false,
   Boolean useVectorFont = false,
   IPresSpaceHandle hpsInit = 0);
```

You can construct objects of this class using the following:

- A window's font. This constructs a font with the same font being in the window.

  Note: In OS/2, if the IWindow* is a multiple-line edit field (MLE), Presentation Manager returns the desktop presentation space instead of the MLE presentation space. In doing so, a different font is displayed.

- A specific font.
- A presentation space. This constructs a font corresponding to the font used in the specified presentation space.
- A copy of another font. This constructs a font with the same specifications as the specified font.

The parameters for these constructors are the following:

window If you do not specify a window, the system default font is used. In the AIX release, if you do not specify a window, the default font of the current display is used.

faceName The face name of the font you want.

ulPointSize
The specific point size you want for the font.

useFixedFont
Specify whether you want the font fixed or proportional. The default is proportional.

useVectorFont
Specify whether you want a vector or bit-map font. The default is a bit-map font.

hpsInit The presentation space handle.

presSpace The presentation space handle.

fntCopy The font you want to copy.

Public Members

avgCharWidth
```
unsigned long avgCharWidth() const;
```

Returns the average character's width.

avgLowercase
```
unsigned long avgLowercase() const;
```

Returns the nominal height above the baseline for lowercase characters.

avgUppercase
```
unsigned long avgUppercase() const;
```

Returns the height of the Em square. This corresponds to the point size.

beginUsingFont
```
virtual IFont &beginUsingFont(
                const IPresSpaceHandle &presSpaceHandle);
```

Sets the presentation space to use the font.

charWidth
```
unsigned long charWidth(char c) const;
```

Returns the width of a specific single-byte character.

c A character. You can only specify SBCS characters.

Motif Information: This member function is not MBCS-enabled.

Presentation Manager Information: This member function is not DBCS-enabled.

endUsingFont

```
virtual IFont &endUsingFont(
                  const IPresSpaceHandle &presSpaceHandle);
```

Restores the presentation space to the default font.

externalLeading

```
unsigned long externalLeading() const;
```

Returns the amount of white space that should appear between adjacent rows of text.

faceNameAt

Condition: (2) OS/2 Only

```
static IString faceNameAt(
                 const FaceNameCursor &faceNameCursor);
```

Queries a font face name for the current position of the specified cursor.

fattrs

Condition: (2) OS/2 Only

```
const struct _FATTRS *fattrs() const;
```

Returns a pointer to the font's OS/2 Presentation Manager FATTRS structure.

fontmetrics

Condition: (2) OS/2 Only

```
const struct _FONTMETRICS *fontmetrics() const;
```

Returns a pointer to the font's OS/2 Presentation Manager FONTMETRICS structure.

internalLeading

```
unsigned long internalLeading() const;
```

Returns the approximate position of the top of a row of characters. You can use this value to position the first line of a block of text by:

1. Subtracting the internalLeading value from the value returned by maxAscender (see page 315).
2. Positioning the baseline of the first line of text below the item (such as text or graphics) that is above the text. Use the value you obtained in step 1 to do so.

Note: There is no guarantee that the positioned characters will not write over the item that is above them. Test to see if anything is over-written and allocate additional space, if necessary.

isBitmap
```
Boolean isBitmap() const;
```

If the IFont object uses a bit-map font, true is returned. If the IFont object uses a vector font, false is returned.

isBitmapOnly
```
Boolean isBitmapOnly() const;
```

If the IFont object uses only bit-map fonts, true is returned.

isBold
```
Boolean isBold() const;
```

If the IFont object uses a font that is bold, true is returned.

isFixed
```
Boolean isFixed() const;
```

If the IFont object uses a fixed-size (that is, non-proportional) font, true is returned.

isItalic
```
Boolean isItalic() const;
```

If the IFont object uses an italicized font, true is returned.

isNonPropOnly
```
Boolean isNonPropOnly() const;
```

If the IFont object uses only non-proportional fonts, true is returned.

isOutline
Condition: (1) Ignored on AIX

```
Boolean isOutline() const;
```

If the IFont object uses a font that appears hollow, true is returned.

isStrikeout
Condition: (1) Ignored on AIX

```
Boolean isStrikeout() const;
```

If the IFont object uses a font with a line drawn through the characters, true is returned.

isUnderscore
Condition: (1) Ignored on AIX

```
Boolean isUnderscore() const;
```

If the IFont object uses a font with a line drawn under the characters, true is returned.

isVectorOnly
```
Boolean isVectorOnly() const;
```

If the IFont object uses only vector fonts, true is returned.

maxAscender
```
unsigned long maxAscender() const;
```

Returns the height of the largest ascender above the baseline.

maxCharHeight
```
unsigned long maxCharHeight() const;
```

Returns the height portion of IFont::maxSize (see page 315).

maxDescender
```
unsigned long maxDescender() const;
```

Returns the height of the largest descender below the baseline.

maxLowercaseAscender
```
unsigned long maxLowercaseAscender() const;
```

Returns the height of the largest lowercase ascender.

maxLowercaseDescender
```
unsigned long maxLowercaseDescender() const;
```

Returns the height of the largest lowercase descender.

maxSize
```
ISize maxSize() const;
```

Returns the maximum width and height.

maxUppercaseSize
```
ISize maxUppercaseSize() const;
```

Returns the maximum size for an uppercase character.

minTextWidth
```
unsigned long minTextWidth(const char *line) const;
```

Returns the width of the widest word.

line Pointer to character string. You can only specify an SBCS string.

Motif Information: This member function is not MBCS-enabled.

Presentation Manager Information: This member function is not DBCS-enabled.

name
```
IString name() const;
```

Returns the face name of the font.

operator =
```
IFont &operator =(const IFont &rhs);
```

Assigns the value of one font object to another.

pointSize
```
unsigned long pointSize() const;
```

Returns the point size of the font.

pointSizeAt
Condition: (2) OS/2 Only

```
static long pointSizeAt(
             const PointSizeCursor &pointSizeCursor);
```

Queries the point size of a font for the current position of the specified cursor.

setAllEmphasis
```
virtual IFont &setAllEmphasis(Boolean turnOn = true);
```

If you specify true, all of the styles are set on. If you specify false, all of the styles are set off.

setBold
```
virtual IFont &setBold(Boolean bold = true);
```

Changes to bold font.

setCharHeight
Condition: (1) Ignored on AIX

```
virtual IFont &setCharHeight(
                 unsigned long height,
                 const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the height of the characters of a vector font. This function has no effect when applied to a bit-map font.

setCharSize

Condition: (1) Ignored on AIX

```
virtual IFont &setCharSize(
                  const ISize &size,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the size of the characters of a vector font. This function has no effect when applied to a bit-map font.

setCharWidth

Condition: (1) Ignored on AIX

```
virtual IFont &setCharWidth(
                  unsigned long width,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the width of the characters of a vector font. This function has no effect when applied to a bit-map font.

setDirection

Condition: (1) Ignored on AIX

```
virtual IFont &setDirection(Direction direction);
```

Sets the direction to draw the font in.

direction Use the enumeration Direction (see page 320) to specify the direction.

Motif Information: In the AIX release, the system locale information determines the font direction.

setFontAngle

Condition: (1) Ignored on AIX

```
virtual IFont &setFontAngle(
                  const IPoint &point,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the angle to draw the vector font with. This function has no effect when applied to a bit-map font.

setFontShear

Condition: (1) Ignored on AIX

```
virtual IFont &setFontShear(
                  const IPoint &point,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the amount of shear to apply to the vector font. This function has no effect when applied to a bit-map font.

setItalic

```
virtual IFont &setItalic(Boolean italics = true);
```

Changes to italic font.

setName

```
virtual IFont &setName(
                  const char *name,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the font's face name.

setOutline

Condition: (1) Ignored on AIX

```
virtual IFont &setOutline(Boolean outline = true);
```

Causes the font to appear hollow.

setPointSize

```
virtual IFont &setPointSize(
                  unsigned long size,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets the font's point size.

setStrikeout

Condition: (1) Ignored on AIX

```
virtual IFont &setStrikeout(Boolean strikeout = true);
```

Draws a line through the characters.

setUnderscore

Condition: (1) Ignored on AIX

```
virtual IFont &setUnderscore(Boolean underscore = true);
```

Draws a line under the characters.

setWindowFont

```
virtual Boolean setWindowFont(IWindow *window) const;
```

Sets the font for the specified IWindow (see page 924) to the current font. If the font is successfully changed, true is returned.

subscriptOffset

```
ISize subscriptOffset() const;
```

Returns the recommended size of the baseline X-Y offset for subscripts.

subscriptSize

```
ISize subscriptSize() const;
```

Returns the recommended size for subscripts.

superscriptOffset

```
ISize superscriptOffset() const;
```

Returns the recommended size of the baseline X-Y offset for superscripts.

superscriptSize

```
ISize superscriptSize() const;
```

Returns the recommended size for superscripts.

textLines

```
unsigned long textLines(
               const char *text,
               unsigned long lineWidth) const;
```

Returns the number of lines required to fit the specified text using the specified maximum line width.

text Pointer to character string. You can only specify an SBCS string.

lineWidth The maximum line width.

Motif Information: This member function is not MBCS-enabled.

Presentation Manager Information: This member function is not DBCS-enabled.

textWidth

```
unsigned long textWidth(const char *text) const;
```

Returns the width of the specified string. **Warning:** This member function is not DBCS-enabled.

text Pointer to character string. You can only specify an SBCS string.

Motif Information: This member function is not MBCS-enabled.

Presentation Manager Information: This member function is not DBCS-enabled.

useBitmapOnly

```
virtual IFont &useBitmapOnly(
                  Boolean bitmapOnly = true,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets whether the IFont object uses only bit-map fonts.

useNonPropOnly

```
virtual IFont &useNonPropOnly(
                  Boolean nonProportionalOnly = true,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets whether the IFont object uses only nonproportional fonts.

useVectorOnly

```
virtual IFont &useVectorOnly(
                  Boolean vectorOnly = true,
                  const IPresSpaceHandle &presSpaceHandle = IPresSpaceHandle ( ));
```

Sets whether the IFont object uses only vector fonts.

Figure 101. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

Direction

```
public:
enum Direction
  {
  defaultDir,
  leftToRight,
  topToBottom,
  rightToLeft,
  bottomToTop,
  leftRight = leftToRight,
  topBottom = topToBottom,
  rightLeft = rightToLeft,
  bottomTop = bottomToTop,
  };
```

Use these enumerators to specify the direction in which a font is drawn. You can use these enumerators as input to setDirection (see page 317).

defaultDir
Draws the font in the default direction.

leftRight
Draws the font from left to right.

topBottom
Draws the font from top to bottom.

rightLeft
Draws the font from right to left.

bottomTop
Draws the font from bottom to top.

Note: The following enumerators are obsolete, use their replacements instead:

Obsolete	Replacement
bottomTop	bottomToTop
leftRight	leftToRight
rightLeft	rightToLeft
topBottom	topToBottom

Chapter 72. IFontDialog

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow
          IFontDialog
```

Header File: ifontdlg.hpp

Objects of the IFontDialog class display a font dialog for the user to choose a font. Once the user chooses a font, you can use accessor functions to retrieve information about the chosen font.

The easiest way to use this class is to specify an IFont (see page 310) object on the IFontDialog::Settings constructor (see page 329). This causes the following to happen:

- The specified IFont is used as the initially displayed font choice in the dialog.
- If the user ends the dialog by selecting the **OK** push button, the specified IFont object is automatically changed to the font the user selected.

Nested Classes

IFontDialog::Style
IFontDialog::Settings

Constructors

```
public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   const Style &style = defaultStyle ( ));

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   const Settings &settings,
   const Style &style = defaultStyle ( ));

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   const Style &style,
   const Settings &settings);
```


© Copyright IBM Corp. 1994

```
public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   IHandler *handler,
   const Style &style = defaultStyle ( ),
   const Settings &settings = Settings ( 0 ));

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   Style style = defaultStyle ( ));

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   const Settings &settings,
   Style style = defaultStyle ( ));

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   Style style,
   const Settings &settings);

public:
IFontDialog(
   IWindow *parent,
   IWindow *owner,
   IHandler *handler,
   Style style = defaultStyle ( ),
   const Settings &settings = Settings ( 0 ));
```

You can construct objects of this class by specifying the following:

- A parent and owner, and optionally, a IFontDialog::Style (see page 332), and IFontDialog::Settings (see page 329). You can specify the Style and Settings in any order following the parent and owner.

  Note: All of these constructors create a modal dialog by default. To construct a modeless dialog, specify the modeless style for *style*.

  In OS/2 Presentation Manager, you can have modal windows that do not use the desktop as their parent. However, Presentation Manager can change the owner window of your dialog to something other than what you specified to prevent your application from being hung. See the *Presentation Manager Programming Reference Volume 2* for more information under WinLoadDlg.

  To ensure your dialog is modal, use the desktop as *parent* and an application window for *owner*. Presentation Manager only disables the owner window and its child windows (the child windows of the child windows, and so forth) while a modal dialog is displayed. If you specify 0 for *parent*, the desktop automatically becomes the parent of the dialog.

- A pointer to an IHandler (see page 402) object. Use this constructor if you want to add an IFontDialogHandler (see page 334) to a modal font dialog. You must

specify the handler on the constructor because once a modal font dialog is created, the dialog does not return control to the program until the user ends the dialog. At that time it is too late to add a handler.

Public Members

buttonPressedId
```
unsigned long buttonPressedId() const;
```

Returns the ID of the push button used to end the dialog.

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

Returns the default style. The default style is classDefaultStyle (see page 332) unless you have changed it using setDefaultStyle (see page 325).

emHeight
```
unsigned long emHeight() const;
```

Returns the height of the Em square for the font.

externalLeading
```
unsigned long externalLeading() const;
```

Returns the amount of white space between lines of text.

fontFamily
```
IString fontFamily() const;
```

Returns the font's family name.

fontWeight
Condition: (1) Ignored on AIX

```
unsigned long fontWeight() const;
```

Returns the weight class (boldness) of the font.

Motif Information: This function always returns 0.

fontWidth
Condition: (1) Ignored on AIX

```
unsigned long fontWidth() const;
```

Returns the width class of the font.

Motif Information: This function always returns 0.

isModeless
```
Boolean isModeless() const;
```

If the font dialog is modeless, true is returned.

nominalPointSize
```
unsigned long nominalPointSize() const;
```

For a bit-map font, the height of the font is returned. For a vector font, the optimal size for the font is returned.

Motif Information: For scalable fonts, this function returns 0.

pointSize
```
unsigned long pointSize() const;
```

Returns the font's point size.

pressedOK
```
Boolean pressedOK() const;
```

If the user ended the dialog by selecting the **OK** push button, true is returned.

returnValue
```
long returnValue() const;
```

If an error occurs, the return code is returned.

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

Sets the default style for all subsequent font dialogs.

style Use the styles provided by IFontDialog::Style (see page 332) to specify the default style.

xHeight
```
unsigned long xHeight() const;
```

Returns the height above the baseline for lowercase letters.

Figure 102. Inherited Public Members

Member function	Class
activeColor	IWindow
addExtension	IFrameWindow
addToWindowList	IFrameWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow

Figure 102. Inherited Public Members

Member function	Class
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IFrameWindow
beginFlashing	IFrameWindow

Figure 102. Inherited Public Members

Member function	Class
borderColor	IWindow
borderHeight	IFrameWindow
borderSize	IFrameWindow
borderWidth	IFrameWindow
characterSize	IWindow
childAt	IWindow
client	IFrameWindow
clientHandle	IFrameWindow
clientRectFor	IFrameWindow
close	IFrameWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IFrameWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
dismiss	IFrameWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IFrameWindow
enableUpdate	IWindow
endFlashing	IFrameWindow
font	IWindow
foregroundColor	IWindow
frameRectFor	IFrameWindow
handle	IWindow
handleException	IWindow
handleFor	IFrameWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
icon	IFrameWindow
id	IWindow
inactiveColor	IWindow
isAnExtension	IFrameWindow

Figure 102. Inherited Public Members

Member function	Class
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFlashing	IFrameWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isMaximized	IFrameWindow
isMinimized	IFrameWindow
isModal	IFrameWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
maximize	IFrameWindow
maximizeRect	IFrameWindow
messageQueue	IWindow
minimize	IFrameWindow
minimizeRect	IFrameWindow
minimumSize	IWindow
mousePointer	IFrameWindow
moveSizeTo	IWindow
moveSizeToClient	IFrameWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
notifyOwner	IFrameWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow

Figure 102. Inherited Public Members

Member function	Class
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeExtension	IFrameWindow
removeFromWindowList	IFrameWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
restore	IFrameWindow
restoreRect	IFrameWindow
result	IFrameWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setBorderHeight	IFrameWindow
setBorderSize	IFrameWindow
setBorderWidth	IFrameWindow
setClient	IFrameWindow
setDestroyOnClose	IFrameWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setExtensionSize	IFrameWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow

Figure 102. Inherited Public Members

Member function	Class
setHiliteForegroundColor	IWindow
setIcon	IFrameWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	IFrameWindow
setMinimumSize	IWindow
setMousePointer	IFrameWindow
setOwner	IWindow
setParent	IWindow
setRestoreRect	IFrameWindow
setResult	IFrameWindow
setShadowColor	IWindow
setToolBarList	IFrameWindow
shadowColor	IWindow
shareParentDBCSStatus	IFrameWindow
show	IWindow
show	IFrameWindow
showModally	IFrameWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
toolBarList	IFrameWindow
topHandle	IWindow
topHandle	IFrameWindow
update	IFrameWindow
useExtensionMinimumSize	IFrameWindow
usesDialogBackground	IFrameWindow
visibleRectangle	IWindow
willDestroyOnClose	IFrameWindow

Note: * Application Support Class Library

Protected Members

Figure 103. Inherited Protected Members

Member function	Class
addDefaultHandler	IFrameWindow
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
attachClient	IFrameWindow
calcMinimumSize	IWindow
create	IWindow
create	IFrameWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
extensions	IFrameWindow
findExtension	IFrameWindow
findExtensions	IFrameWindow
initialize	IFrameWindow
isFrameWindow	IFrameWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
isRelatedHandle	IFrameWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IFrameWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeDefaultHandler	IFrameWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setExtensions	IFrameWindow
setNotificationHandler	IWindow
setStyle	IWindow

Figure 103. Inherited Protected Members

Member function	Class
setWindowData	IWindow
start	IFrameWindow
startHandlingEventsFor	IWindow
style	IWindow
tryToLoadDialog	IFrameWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IFrameWindow
windowSystemClass	IFrameWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 73. IFontDialog::Settings

Derivation

IBase
  IFontDialog::Settings

Header File: ifontdlg.hpp

Objects of the IFontDialog::Settings class contain input for an IFontDialog (see page 322) object. The font dialog displays with the items you specify in the Settings object. If you do not specify a Settings object on the IFontDialog constructor, a default Settings object is constructed for you. This default object provides a font dialog with no other settings in effect. You can pass the following items within an IFontDialog::Settings object to an IFontDialog:

- The dialog title
- The push button arrangement on the dialog
- The characteristics of the fonts to present, such as family or point size

Presentation Manager Information: You can also pass the following items within a Settings object:

- The presentation space from which the dialog generates the font list
- A dialog template to use in place of the system default dialog

Constructors

```
public:
Settings(IFont *font = 0);

public:
Settings(const Settings &another);
```

You can construct objects of this class by using the default constructor, which accepts an optional IFont object. You can specify a font to display as the initial selection in the dialog. If the user selects the **OK** push button, the specified font is changed to the font selected by the user.

Public Members

operator =

```
Settings &operator =(const Settings &another);
```

Sets the dialog font.

© Copyright IBM Corp. 1994

setDialogTemplate

Condition: (2) OS/2 Only

```
Settings &setDialogTemplate(const IResourceId &templateId);
```

Sets a dialog template resource to use in place of the OS/2-supplied default font dialog.

setDisplayPS

Condition: (1) Ignored on AIX

```
Settings &setDisplayPS(
            const IPresSpaceHandle &presSpaceHandle);
```

Sets the display's presentation space. The dialog uses this presentation space to determine which fonts are available.

setFamily

```
Settings &setFamily(const char *fontFamily);
```

Sets the font family name for the initial set of fonts displayed in the font dialog.

setPointSize

```
Settings &setPointSize(unsigned long pointSize);
```

Sets the point size for the initial set of fonts displayed in the font dialog.

Motif Information: If you specify a point size of 0, X-scalable fonts are displayed.

setPosition

```
Settings &setPosition(const IPoint &position);
```

Sets the initial placement of the dialog.

setPreviewText

```
Settings &setPreviewText(const char *previewText);
```

Sets the text to display in the font sample box.

setPrinterPS

Condition: (1) Ignored on AIX

```
Settings &setPrinterPS(
            const IPresSpaceHandle &presSpaceHandle);
```

Sets the display's presentation space. The dialog uses this presentaion space to determine which fonts are available.

setSizeList

```
Settings &setSizeList(const char *sizeList);
```

Sets the list of point size choices.

Motif Information: This function has no effect in Motif.

Presentation Manager Information: This only affects dialogs for vector fonts in an OS/2 system. A dialog for an OS/2 bit-map font has the size list automatically set to the available sizes for the bit-map font.

setTitle

```
Settings &setTitle(const char *title);

Settings &setTitle(const IResourceId &text);
```

Sets the font dialog's title.

Figure 104. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 74. IFontDialog::Style

Derivation

IBase
  IBitFlag
    IFontDialog::Style

Header File: ifontdlg.hpp

The nested class IFontDialog::Style provides a set of valid font dialog styles for the static member functions of the class IFontDialog (see page 322).

Styles

applyButton
Adds an **Apply** push button to the dialog. This is useful in a modeless dialog.

bitmapOnly
Specifies the dialog present bit-map fonts only.

classDefaultStyle
Specifies the original default style for this class, which is a static variable set to 0 indicating no style flags are set.

fixedWidthOnly
Specifies the dialog present fixed-width (monospace) fonts only.

helpButton
Adds a **Help** push button to the dialog. This push button sends a help message to the owner of the dialog.

modeless
Specifies the dialog is modeless. The constructor for a modeless dialog returns immediately. To determine the font chosen by the user, create an IFontDialogHandler and use IFontDialog::modelessResults (see page 335).

noStyle
Disables all of the valid styles.

noSynthesize
Specifies the dialog does not manipulate bit-map fonts to synthesize features, such as italics or bold.

proportionalOnly
Specifies the dialog present proportionally-spaced fonts only.

resetButton
Adds a **Reset** push button to the dialog. When this push button is selected, the values for the dialog are restored to their initial values.

© Copyright IBM Corp. 1994

vectorOnly
Specifies the dialog present vector fonts only.

Motif Information: In Motif, the style vectorOnly specifies the dialog present scalable fonts only. AIX does not support the noSynthesize style.

Public Members

Figure 105. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 106. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 75. IFontDialogHandler

Derivation

```
IBase
  IVBase
    IHandler
      IFontDialogHandler
```

Header File: ifonthdr.hpp

Objects of the IFontDialogHandler class handle the various messages that affect the font dialog.

Create a handler derived from IFontDialogHandler and attach it to a font dialog. You can do this by calling IHandler::handleEventsFor (see page 403) to pass the appropriate font dialog to the font dialog handler.

Note: To add a handler to a modal font dialog, you must specify a pointer to the handler on the IFontDialog (see page 322) constructor.

When the font dialog handler receives a font dialog event, it creates an IFontDialog object and routes that object to the IFontDialogHandler::modelessResults (see page 335) virtual function. Override this virtual function to supply your own specialized processing of a font dialog event.

The return value from the virtual function specifies whether the font dialog event is passed on for additional processing, as follows:

true The font dialog event requires no additional processing. Do not pass it to another handler.

false Pass the font dialog event to the next handler for additional processing, as follows:

- If there is another handler for the font dialog, pass the font dialog event to the next handler.
- If this is the last handler for the font dialog, call IWindow::defaultProcedure (see page 946) to process the font dialog event.

Constructors

```
public:
IFontDialogHandler();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

© Copyright IBM Corp. 1994

Public Members

Figure 107. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a font dialog event is received, the appropriate virtual function is called.

modelessResults

```
virtual Boolean modelessResults(
                  IFontDialog *endingDialog) = 0;
```

Called when a modeless dialog is ended. This function is called even if the dialog is cancelled. This is done so that you can destroy the IFontDialog (see page 322) if needed. Use IFontDialog::pressedOK (see page 325) to determine whether the user ended the dialog by pressing the Ok push button.

If you need to destroy the IFontDialog object, do not delete the IFontDialog* pointer from within IFontDialog::modelessResults. Instead, call:

```
endingDialog->setAutoDeleteObject().
```

Figure 108. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 76. IFrameEvent

Derivation

IBase
  IVBase
    IEvent
      IFrameEvent

Header File: iframevt.hpp

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.

Constructors

```
public:
IFrameEvent(const IEvent &baseEvent);
```

Although you can construct objects of this class, typically IFrameHandler::dispatchHandlerEvent (see page 351) creates objects of this class from an object of the class IEvent (see page 275). The event's window handle must be that of a frame window.

Public Members

frame

```
IFrameWindow *frame() const;
```

Returns a pointer to the frame window receiving this event.

Figure 109. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent

Figure 109. Inherited Public Members

Member function	Class
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent

© Copyright IBM Corp. 1994

Figure 109. Inherited Public Members

Member function	Class
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 77. IFrameExtension

Derivation

IBase
  IFrameExtension

Header File: iframext.hpp

IFrameWindow (see page 352) maintains objects of the IFrameExtension class to record information about each of its frame extension windows. You add a frame window as a frame extension by calling IFrameWindow::addExtension (see page 355). You do not need to create an IFrameExtension object yourself.

IFrameWindow::extensions (see page 370) and IFrameWindow::setExtensions (see page 372) deal with collections of such frame extensions. Each collection is actually an object of the IFrameExtensions class, which is essentially a synonym for ISequence<IFrameExtension*>. The class ISequence is in the IBM Collection Class Library.

Frame extensions have the following attributes:

type Specifies the extension be one of two types having:

- A fixed width or height.
- Width or height that is a fraction of the frame control it occupies. See the location attribute below.

size Specifies the size is either:

- A specific width or height, for fixed-size extensions
- A fraction, for relative-sized extensions

location Specifies the portion of the frame the extension occupies. The enumeration IFrameWindow::Location (see page 374) provides the possible values.

control Specifies a pointer to the control window that occupies the extension.

Constructors

```
public:
IFrameExtension(
   IWindow *window,
   IFrameWindow::Location location,
   unsigned long widthOrHeight,
   IFrameWindow::SeparatorType separator);

public:
IFrameExtension(
   IWindow *window,
   IFrameWindow::Location location,
   double percentage,
   IFrameWindow::SeparatorType separator);
```


© Copyright IBM Corp. 1994

Condition: (2) OS/2 Only

```
public:
IFrameExtension(
   IWindow *window,
   IFrameWindow::Location location,
   IFrameWindow::SeparatorType separator);
```

You can construct objects of this class by providing the same set of parameters as for IFrameWindow::addExtension (see page 355).

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 (see page 374) 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.

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 (see page 375) to specify the type of separator delimiting the frame extension.

Public Members

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

Returns the ID of the frame control this extension is attached to.

attachTo
```
virtual ISWP attachTo(ISWP &baseSWP);
```

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.

baseSWP The position of the control the frame extension is attached to.

Motif Information: AIX does not support this function.

baseRectFor

```
virtual IRectangle baseRectFor(
                    const IRectangle &totalRect) const;
```

Calculates and returns a rectangle representing the client rectangle without its frame extension.

totalRect The total rectangle which is the client rectangle plus the frame extension's size and position.

Motif Information: This function always returns a default IRectangle.

control

```
IWindow *control() const;
```

Returns a pointer to the IControl (see page 220) window that occupies this frame extension.

drawSeparator

```
virtual void drawSeparator(const IPresSpaceHandle &psh);
```

Draws the separator line between a frame extension and the adjacent control (or extension) it is attached to.

psh

fixedSize

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

Returns the fixed width or height in window coordinates. If the extension is of type relative, 0 is returned.

location

```
IFrameWindow::Location location() const;
```

Returns the frame extension's location.

relativeSize

```
virtual double relativeSize() const;
```

Returns the relative width or height as a percentage. If the extension is of type fixed, 0.0 is returned.

separatorHandle

Condition: (3) AIX only

```
virtual IWindowHandle separatorHandle() const;
```

Returns the handle of the separator.

separatorType

Condition: (3) AIX only

```
virtual IFrameWindow::SeparatorType separatorType() const;
```

Returns the separator type for this frame extension object. The returned value is an enumerator provided by IFrameWindow::SeparatorType (see page 375).

separatorWidth

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

Returns the width of the separator line between this frame extension and either:

- The control it is attached to
- The extension it is adjacent to

setSeparatorHandle

Condition: (3) AIX only

```
virtual IFrameExtension &setSeparatorHandle(
                          const IWindowHandle &separatorHandle);
```

Sets the handle of the separator object.

setSize

```
virtual IFrameExtension &setSize(
                          unsigned long widthOrHeight);

virtual IFrameExtension &setSize(double widthOrHeight);
```

Resets the size of the extension.

widthOrHeight
: The portion of the frame window area (title bar area, menu bar area, or client area) allocated to the frame extension, in either

  - A fixed width or height (unsigned long)
  - A floating point percentage (double, 0 < x < 1.0)

sizeTo

```
virtual ISize sizeTo(const ISize &baseSize) const;
```

Calculates and returns the size this extension would be if attached to a control of the specified size.

baseSize The size of the proposed control.

Motif Information: AIX does not support this function. The function always returns the specified baseSize value.

totalRectFor

```
virtual IRectangle totalRectFor(
                    const IRectangle &baseRect) const;
```

Calculates and returns a new rectangle value. This value is *baseRect* plus the frame extension's size and position.

baseRect A rectangle representing the client rectangle without its frame extension.

Motif Information: This function always returns a default IRectangle.

type

```
Type type() const;
```

Returns the extension type for this frame extension object. The returned value is an enumerator provided by Type (see page 342).

useMinimumSize

Condition: (2) OS/2 Only

```
virtual IFrameExtension &useMinimumSize();
```

Specifies that the width or height of the extension is assigned from the minimum size of the window it contains.

Figure 110. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Enumerations

Type

```
public:
enum Type
  {
  fixed,
  relative,
  minimumSize,
  };
```

Use these enumerators to specify the type of frame extension:

fixed
Specifies a fixed width or height.

relative
Specifies a relative width or height, based on how much of the frame control the extension occupies.

Chapter 78. IFrameExtensions

Derivation

ISequence
  IFrameExtensions

Header File: iframext.hpp

The IFrameExtensions class is a wrapper for a set, or sequence, of IFrameExtension (see page 338) objects. The User Interface Class Library uses the IFrameExtensions class to add a level of indirection in the IFrameWindow (see page 352) interface.

The library derives the IFrameExtensions class from the class ISequence which is in the IBM Collection Class Library.

Constructors

```
public:
IFrameExtensions();
```

You can only construct objects of this class from the default constructor which accepts no parameters.

© Copyright IBM Corp. 1994

Chapter 79. IFrameFormatEvent

Derivation

IBase
  IVBase
    IEvent
      IFrameEvent
        IFrameFormatEvent

Header File: iframevt.hpp

Objects of the IFrameFormatEvent class represent information about a frame format event. The frame window handler creates and dispatches objects of this class to IFrameHandler::format (see page 349) when a frame format event occurs.

Motif Information: AIX does not support this class.

Constructors

Condition: (1) Ignored on AIX

```
public:
IFrameFormatEvent(const IEvent &baseEvent);
```

Although you can construct objects of this class, typically IFrameHandler::dispatchHandlerEvent (see page 351) creates objects of this class from an object of the class IEvent (see page 275).

The IEvent object is constructed from the following:

- WM_FORMATFRAME for the event ID.
- The event's first parameter be a pointer to an array of SWP (see page 861) structures. These structures' dimensions must be equal to the frame control count returned on the preceding WM_QUERYFRAMECTLCOUNT event.
- The event's second parameter be a pointer to a RECTL structure in which the client control's size and position are returned.

The event result field ultimately contains the count of the number of frame controls processed.

Public Members

© Copyright IBM Corp. 1994

clientRect

Condition: (1) Ignored on AIX

```
IRectangle clientRect() const;
```

Returns the client rectangle associated with this event.

setClientRect

Condition: (1) Ignored on AIX

```
IFrameFormatEvent &setClientRect(const IRectangle &aRect);
```

Sets the client rectangle.

aRect The client rectangle value.

swpArray

Condition: (1) Ignored on AIX

```
ISWP *swpArray() const;
```

Returns a pointer to the ISWP (see page 861) array associated with this event.

Figure 111. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
frame	IFrameEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent

Figure 111. Inherited Public Members

Member function	Class
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 80. IFrameHandler

Derivation

IBase
  IVBase
    IHandler
      IFrameHandler

Header File: iframhdr.hpp

Objects of the IFrameHandler class handle frame window events. Usually, you create an IFrameHandler object and attach it to the frame at the time of construction using IFrameWindow::start (see page 372). When you use any of the protected member functions IFrameWindow::create (see page 369), IFrameWindow::tryToLoadDialog (see page 373), or IFrameWindow::initialize (see page 371) and you do not specify the style deferCreation (see page 376), 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, create a handler derived from IFrameHandler and attach it to the IFrameWindow object. You can do this by calling IHandler::handleEventsFor (see page 403) to pass the appropriate frame window to the frame window handler.

IFrameHandler dispatches the following events:

- Activates, closes, deactivates, draws, and saves the frame window.
- Receives commands. ICommandEvent (see page 128) discusses this in more detail.
- Formats frame extensions. IFrameFormatEvent (see page 345) 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 (see page 946) to process the frame event.

© Copyright IBM Corp. 1994

Note: Because IFrameHandler processes ICommandEvents, you must think of it as if you had more than one ICommandHandler attached to a window. Additionally, if you attach an IFrameHandler to the same window as an ICommandHandler, your ICommandHandler might not be called depending on the order that you attach these handlers.

Portability Considerations: The AIX environment does not support:

IFrameHandler::calcRect
IFrameHandler::deactivated
IFrameHandler::format

Constructors

```
public:
IFrameHandler();
```

You can only construct objects of this class using the default constructor, which accepts no parameters.

Public Members

activated

```
virtual Boolean activated(IFrameEvent &frameEvent);
```

Activates the frame window. When the frame handler receives a frame event object containing the event ID for activating the frame window, this function's implementation does not do anything. However, derived classes can implement this function by:

- Performing their event processing
- Returning false so other handlers can act on this event

Motif Information: This function is called when the WM_TAKE_FOCUS message is received from the Motif Window Manager.

calcRect

```
virtual Boolean calcRect(IFrameEvent &frameEvent);
```

Calculates either the size of:

- The client rectangle given a frame rectangle
- The frame rectangle given a client rectangle

When the frame handler receives a frame event object containing the event ID for calculating either of these rectangles, this function performs the calculations.

Motif Information: AIX does not support this function. The function always returns false.

closed

```
virtual Boolean closed(IFrameEvent &frameEvent);
```

Closes the frame window. When the frame handler receives a frame event object containing the event ID for closing the frame window, this function:

1. Queries whether the frame is modal
2. If so, it dismisses the frame
3. Queries the state of the destroyOnClose flag maintained by the IFrameWindow (see page 352)
4. If this flag is on, it destroys the window
5. If the flag is off and there is a client, it posts an SC_CLOSE systemCommand to the client.

This function always returns true.

command

```
virtual Boolean command(ICommandEvent &cmdEvent);
```

Processes a command event. If the frame has a client, false is returned. This lets the client receive the event. Otherwise, true is returned.

deactivated

Condition: (1) Ignored on AIX

```
virtual Boolean deactivated(IFrameEvent &frameEvent);
```

Deactivates the frame window. When the frame handler receives a frame event object containing the event ID for giving up the input focus, this function's implementation does not do anything. However, derived classes can implement this function by:

- Performing their event processing
- Returning false so other handlers can act on this event

draw

Condition: (1) Ignored on AIX

```
virtual Boolean draw(IFrameEvent &frameEvent);
```

Redraws the frame window. When the frame handler receives a frame event object containing the event ID for redrawing the frame, this function draws the separator lines between frame extensions and returns false. Derived classes can implement their own version of this function by:

- Performing any additional drawing they require
- Returning false so other handlers can act on this event

format

Condition: (1) Ignored on AIX

```
virtual Boolean format(IFrameFormatEvent &formatEvent);
```

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 (see page 355). Derived classes can implement this function by:

1. Formatting their controls
2. Returning true, so the standard frame controls are not formatted again

numberOfExtensions

```
virtual unsigned numberOfExtensions(
                  const IFrameWindow *pFrame) const;
```

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 (see page 370) function.

saved

```
virtual Boolean saved(IFrameEvent &frameEvent);
```

Saves the application status. When the frame handler receives a frame event object containing the event ID for saving the status, this function's implementation does not do anything except return false. However, derived classes can implement this function by:

- Saving the application status
- Returning false so other handlers can act on this event

Figure 112. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

```
virtual Boolean dispatchHandlerEvent(IEvent &anEvent);
```

If a frame event is received, the appropriate virtual function is called.

anEvent The received frame event.

Motif Information: 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.

positionExtensions

Condition: (1) Ignored on AIX

```
virtual unsigned long positionExtensions(
                      IFrameFormatEvent &formatEvent,
                      unsigned long numStdControls);
```

Positions the frame of extensions after you have altered the size or position of a standard frame control. You should call this function from within the format function after you have modified the ISWP objects for a standard frame control. Return true from format to prevent the default frame handler from positioning the frame extensions again.

Figure 113. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 81. IFrameWindow

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow
```

Header File: iframe.hpp

Objects of the IFrameWindow class create and manage 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 (see page 924) 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 the other frame controls. 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 (see page 441) 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 the IWindow will occupy. The extension can fill a portion of the standard title-bar area (the left- or right-hand side of a specified width) or the standard client area (the top or bottom edge of a specified height).

Portability Considerations: The AIX release of the User Interface Class Library:

- Does not support dialog templates.

  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 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 will not overlay this window (that is, the secondary frame window will be raised).

Motif Information: 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:

© Copyright IBM Corp. 1994

- A window menu button
- A title bar
- Minimize and maximize buttons
- A border
- A menu bar
- A client

If you set the titleBar 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. A XmMainWindow and a XmForm provide the menubar support and client area. Frame extensions are supported through attachments on the XmForm widget.

Presentation Manager Information: 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 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.

Nested Classes

IFrameWindow::Style

Constructors

```
public:
IFrameWindow(unsigned long id = IC_DEFAULT_FRAME_ID);

public:
IFrameWindow(const IResourceId &resId, IWindow *owner = 0);
```

```
public:
IFrameWindow(
   const IResourceId &resId,
   IWindow *parent,
   IWindow *owner);

public:
IFrameWindow(const IWindowHandle &hwnd);

public:
IFrameWindow(
   const IResourceId &resId,
   const IFrameWindow::Style &style);

public:
IFrameWindow(
   const IFrameWindow::Style &style,
   const IResourceId &resId = IC_DEFAULT_FRAME_ID);

public:
IFrameWindow(
   const char *title,
   const IResourceId &resId = IC_DEFAULT_FRAME_ID,
   const IFrameWindow::Style &style = defaultStyle ( ));

public:
IFrameWindow(
   const IResourceId &resId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initRect,
   const IFrameWindow::Style &style = defaultStyle ( ),
   const char *title = 0);
```

The first three constructors use dialog templates to create windows.

- From a specified window ID.

  Constructs a frame window using the dialog template resource with the specified ID. If no such template is found, the constructor creates a default frame window and gives it the specified ID.

- From a specified resource ID and, optionally, an owner window pointer.

  Constructs either primary or secondary frame windows. You must specify a resource ID. The owner window pointer defaults to 0. Therefore, to construct a secondary window, you must specify *owner*.

- From a specified resource ID, parent window, and owner window.

  Constructs a child window. You must specify the resource ID, parent, and owner of the child window.

The remaining constructors do not create windows from dialog templates. Instead, you specify frame attributes as additional constructor parameters.

- From the window handle of an existing frame window.

  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.

Note: This constructor is not supported in AIX.

- 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.

- 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_ that is defined in <icconst.h>.

- 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_ that is defined in <icconst.h>. If you do not specify *style*, the constructor calls defaultStyle (see page 358) to return the default style.

- 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 non-dialog frame windows. If you do not specify *style*, the constructor calls defaultStyle (see page 358) to return the default style.

  The default for *title* is 0. If you set the style titleBar (see page 376), 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.

Public Members

addExtension

```
virtual IFrameWindow &addExtension(
                        IWindow *aWindow,
                        Location aLocation,
                        double percentage,
                        SeparatorType separator = thinLine);

virtual IFrameWindow &addExtension(
                        IWindow *aWindow,
                        Location aLocation,
                        unsigned long widthOrHeight,
                        SeparatorType separator = thinLine);
```

Condition: (2) OS/2 Only

```
virtual IFrameWindow &addExtension(
                        IWindow *aWindow,
                        Location aLocation,
                        SeparatorType separator = thinLine);
```

Adds a window as a frame extension.

aWindow The frame extension window.

aLocation 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 (see page 374) 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.

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 SeparatorType (see page 375) to specify the type of separator delimiting the frame extension.

addToWindowList

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &addToWindowList();
```

Adds this frame window's title as a window list entry. If you construct this frame window with the style windowList (see page 376), the title is automatically added to the window list.

backgroundColor

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

Returns the color of the frame window background.

beginFlashing

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &beginFlashing();
```

Starts flashing the frame. Use this to alert the user to something that requires immediate attention.

borderHeight

Condition: (1) Ignored on AIX

```
unsigned long borderHeight() const;
```

Returns the height of the frame window's top and bottom borders.

borderSize

Condition: (1) Ignored on AIX

```
ISize borderSize() const;
```

Returns the width of the frame window's left and right borders and the height of the frame window's top and bottom borders.

Motif Information: In AIX, this function always returns ISize(1,1).

borderWidth

Condition: (1) Ignored on AIX

```
unsigned long borderWidth() const;
```

Returns the width of the frame window's left and right borders.

client

```
virtual IWindow *client() const;
```

Returns a pointer to the IWindow (see page 924) corresponding to the client window. If no client window exists, 0 is returned.

clientHandle

```
virtual IWindowHandle clientHandle() const;
```

Returns the IWindowHandle (see page 960) of the client window. If no client window exists, a NULL handle is returned.

clientRectFor

```
virtual IRectangle clientRectFor(
                    const IRectangle &frameRect) const;
```

Returns a rectangle indicating the size and position the client window has if the frame is sized and positioned according to the specified rectangle.

frameRect The proposed size and position of the frame.

close

```
virtual IFrameWindow &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.

defaultOrdering

```
static IWindow::SiblingOrder defaultOrdering();
```

Returns the order in which new windows are created relative to their sibling windows.

defaultStyle

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

Returns the default style. This style is classDefaultStyle (see page 376) unless you have changed the style using setDefaultStyle (see page 363).

disabledBackgroundColor

```
virtual IColor disabledBackgroundColor() const;
```

Returns the color of the frame window background when the window is disabled.

dismiss

```
virtual IFrameWindow &dismiss(unsigned long result = 0);
```

Closes a frame window by hiding it and, if modal, completes the calls to IFrameWindow::showModally (see page 366) that triggered the window's display.

result Value for this use of the frame window. The default is 0.

enableNotification

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

Enables or disables event notifications from the frame window to any interested observer objects.

enable Enablement flag. If true, notification is enabled; if false, notification is disabled.

endFlashing

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &endFlashing();
```

Stops flashing the frame window.

frameRectFor

```
virtual IRectangle frameRectFor(
                    const IRectangle &clientRect) const;
```

Returns a rectangle indicating the frame window size and position required to produce the specified client window.

clientRect The proposed client window size and position.

handleFor

```
virtual IWindowHandle handleFor(
                       const Style &aStdControl) const;
```

Returns the IWindowHandle (see page 960) corresponding to a standard frame control. If the control is not present, 0 is returned.

aStdControl
The specified standard control. The parameter corresponds to one of the Presentation Manager standard frame controls.

Motif Information: In AIX, this function always returns 0.

icon

```
virtual IPointerHandle icon() const;
```

Returns the pointer handle of the window icon.

isAnExtension

Condition: (3) AIX only

```
virtual Boolean isAnExtension(IWindow *aWindow);
```

Returns true if the specified IWindow object is a frame extension.

isFlashing

Condition: (1) Ignored on AIX

```
Boolean isFlashing() const;
```

If the frame window is flashing, true is returned.

Motif Information: In AIX, this function always returns false.

isMaximized

Condition: (1) Ignored on AIX

```
Boolean isMaximized() const;
```

If the frame window is maximized, true is returned.

Motif Information: In AIX, this function always returns false.

isMinimized

```
Boolean isMinimized() const;
```

If the frame window is minimized, true is returned.

isModal

```
Boolean isModal() const;
```

If the window is displayed in application-modal mode, true is returned.

maximize

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &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.

maximizeRect

Condition: (1) Ignored on AIX

```
virtual IRectangle maximizeRect() const;
```

Returns a rectangle indicating the size and position the frame window has when it is maximized.

Motif Information: AIX does not support this function, therefore it returns a default IRectangle.

minimize

```
virtual IFrameWindow &minimize();
```

Minimizes the window.

minimizeRect

Condition: (1) Ignored on AIX

```
virtual IRectangle minimizeRect() const;
```

Returns a rectangle indicating the size and position the frame window has when it is minimized.

Motif Information: AIX does not support this function, therefore it returns a default IRectangle.

mousePointer

Condition: (2) OS/2 Only

```
virtual IPointerHandle mousePointer() const;
```

Returns the handle of the mouse pointer used when the mouse is over the frame window. If the handle is null, the default mouse pointer is used.

moveSizeToClient

```
virtual IFrameWindow &moveSizeToClient(
                        const IRectangle &clientRect);
```

Sizes and positions the frame window around the specified client window rectangle.

clientRect The client window rectangle for the frame window to contain.

nextShellRect

Condition: (1) Ignored on AIX

```
static IRectangle nextShellRect();
```

Returns the system-recommended coordinates of the next main window.

Motif Information: 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.

notifyOwner

```
virtual IFrameWindow &notifyOwner(
                        unsigned long id,
                        ICommandEvent::Source source = ICommandEvent::unknown,
                        Boolean pointerDevice = false);
```

Posts a command event to the frame window's (or dialog's) owner.

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 (see page 130) to specify the origin of the command event. The default is the enumerator ICommandEvent::unknown, which indicates control-specific command information.

ptrDevice 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.

removeExtension

```
virtual IFrameWindow &removeExtension(IWindow *aWindow);
```

Removes a frame extension (window).

removeFromWindowList

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &removeFromWindowList();
```

Removes this frame's entry from the window list.

restore

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &restore();
```

Restores a maximized or minimized window to its previous size and position.

Motif Information: This function only restores a window from its minimized state.

restoreRect

Condition: (1) Ignored on AIX

```
virtual IRectangle restoreRect() const;
```

Returns the window rectangle coordinates indicating the size and position the window will be restored to.

Motif Information: This function returns a default IRectangle object.

result

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

Returns the frame result value. You set this value previously by calling either IFrameWindow::setResult (see page 365) or IFrameWindow::dismiss (see page 358).

setBorderHeight

Condition: (1) Ignored on AIX

```
IFrameWindow &setBorderHeight(unsigned long cy);
```

Sets the height of the frame window's top and bottom borders.

cy The height of the top and bottom borders.

setBorderSize

Condition: (1) Ignored on AIX

```
IFrameWindow &setBorderSize(const ISize &size);
```

Condition: (1) Ignored on AIX

```
IFrameWindow &setBorderSize(unsigned long cxcy);
```

Condition: (1) Ignored on AIX

```
IFrameWindow &setBorderSize(
                unsigned long cx,
                unsigned long cy);
```

Sets the width of the frame window's left and right borders and the height of the frame window's top and bottom borders.

cx The width of the left and right borders.

cy The height of the top and bottom borders.

size The width of the left and right borders and the height of the top and bottom borders.

cxcy The uniform size of all borders.

setBorderWidth

Condition: (1) Ignored on AIX

```
IFrameWindow &setBorderWidth(unsigned long cx);
```

Sets the width of the frame window's left and right borders.

cx The width of the left and right borders.

setClient

```
virtual IFrameWindow &setClient(IWindow *newClient);
```

Specifies the IWindow (see page 924) corresponding to the frame window's client area.

newClient The window to set as the frame window's client.

setDefaultOrdering

```
static void setDefaultOrdering(
            IWindow::SiblingOrder ordering);
```

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.

ordering Use the enumeration IWindow::SiblingOrder (see page 953) to specify the order of sibling window creation. If you specify onTopOfSiblings, the library creates new windows on top of their siblings.

setDefaultStyle

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

Sets the default style for all subsequent frame windows.

style Use the styles provided by IFrameWindow::Styles (see page 376) to specify the default style.

setDestroyOnClose

```
virtual IFrameWindow &setDestroyOnClose(
                        Boolean destroy = true);
```

Sets the destroy window flag on or off. If the flag is on, the window object is destroyed when the window is closed.

destroy The destroy window flag value. The default value is true (on).

setExtensionSize

```
virtual IFrameWindow &setExtensionSize(
                        IWindow *aWindow,
                        double widthOrHeight);

virtual IFrameWindow &setExtensionSize(
                        IWindow *aWindow,
                        unsigned long widthOrHeight);
```

Sets a frame extension to the specified size.

aWindow 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 either:

- A fixed width or height (unsigned long).
- A floating point percentage (double, 0 < x < 1.0).

setIcon

```
virtual IFrameWindow &setIcon(const IPointerHandle &icon);

virtual IFrameWindow &setIcon(const IResourceId &iconResId);

virtual IFrameWindow &setIcon(unsigned long iconResId);
```

Sets the window icon.

icon The pointer handle of the window icon.

iconResId The resource identifier of the window icon.

Motif Information: 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 will scale the icon to 50x50. Since scaling can cause a significant loss of resolution, keep the icon size as close to 50x50 as possible.

setLayoutDistorted

Condition: (2) OS/2 Only

```
IFrameWindow &setLayoutDistorted(
                unsigned long layoutAttributesOn,
                unsigned long layoutAttributesOff);
```

Indicates that changes have occurred in the window that will cause the layout of the window to be updated.

setMousePointer

Condition: (2) OS/2 Only

```
virtual IFrameWindow &setMousePointer(
                        const IPointerHandle &mousePointer);
```

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.

setRestoreRect

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &setRestoreRect(
                        const IRectangle &rect);
```

Sets the rectangle coordinates indicating the size and position to restore the window to.

aRect The rectangle coordinates to restore the window to.

setResult

```
virtual IFrameWindow &setResult(unsigned long result);
```

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 (see page 362). For example:

```
IFrameWindow dialog( MY_DIALOG );
dialog.showModally();
if ( dialog.result() == DID_OK )
  // User said to go ahead...
else
  // User changed his mind...
```

result The result value for this use of the frame window.

setToolBarList

Condition: (2) OS/2 Only

```
IFrameWindow &setToolBarList(IToolBarList *toolBarList);
```

Stores the address of a list containing the tool bars used in this frame window.

shareParentDBCSStatus

Condition: (1) Ignored on AIX

```
virtual IFrameWindow &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 appDBCSStatus (see page 376) set, this function has no effect.

show

```
virtual IFrameWindow &show(Boolean showWindow = true);
```

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.

showModally

```
virtual unsigned long showModally();
```

Displays the frame window in application-modal mode (all other application windows are disabled).

toolBarList

Condition: (2) OS/2 Only

```
IToolBarList *toolBarList() const;
```

Returns the address of the list of tool bars used in this frame window.

topHandle

Condition: (3) AIX only

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

Returns the top (the oldest) window system control of the IFrameWindow object.

update

```
virtual IFrameWindow &update();
```

Reconfigures all of the controls and extensions for the frame window. The functions addExtension (see page 355) and setClient (see page 363) automatically call this function.

useExtensionMinimumSize

Condition: (2) OS/2 Only

```
virtual IFrameWindow &useExtensionMinimumSize(
                        IWindow *aWindow);
```

Sizes the width or height of the frame extension based on the minimum size of the window it contains.

usesDialogBackground

Condition: (1) Ignored on AIX

```
Boolean usesDialogBackground() const;
```

Returns whether the frame window uses the system dialog background color rather than the system window background color.

Motif Information: This function returns false.

willDestroyOnClose

```
Boolean willDestroyOnClose() const;
```

Returns the state of the destroy window flag. The default state is on.

Figure 114. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow

Figure 114. Inherited Public Members

Member function	Class
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow

Figure 114. Inherited Public Members

Member function	Class
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow

Figure 114. Inherited Public Members

Member function	Class
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

addDefaultHandler

```
IFrameWindow &addDefaultHandler();
```

Adds the default frame handler to a newly created frame.

attachClient

Condition: (3) AIX only

```
IFrameWindow &attachClient(IFrameExtension *p);
```

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.

create

Condition: (3) AIX only

```
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 ( ));
```

Condition: (2) OS/2 Only

```
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 ( ));
```

Creates a frame window.

id The frame window resource ID.

title The frame window title.

style The default style specification for the frame window. Use the styles provided by IFrameWindow::Styles (see page 376) to specify the default style.

windowClass
The window class of the frame window (OS/2 only).

createFunction
The create widget function for the widget type (AIX only).

parent The parent of the frame window.

owner The owner of the frame window.

initRect The initial frame window rectangle, defined by IFrameWindow::nextShellRect (see page 361).

ctlData Control data for the window (OS/2 only).

presParams
Presentation parameters for the window (OS/2 only).

callerArgList
The resource argument list for the widget type (AIX only).

callerNumberArguments
The number of resource arguments in the *callerArgList* (AIX only).

ordering Sibling window order. Use the enumeration IWindow::SiblingOrder (see page 953) to specify the order of sibling window creation.

extensions

```
IFrameExtensions *extensions() const;
```

Returns a pointer to the IFrameExtensions (see page 344) collection for the frame window.

findExtension

```
unsigned findExtension(IWindow *window);
```

Returns the index of the frame extension corresponding to the specified control IWindow (see page 924). The index locates the IFrameExtension (see page 338) in the IFrameExtensions (see page 344) collection of the frame window.

window The window associated with the frame extension.

findExtensions

Condition: (3) AIX only

Condition: (4) Typically for internal use

```
IFrameWindow &findExtensions(
                IFrameExtension * &topExt,
                IFrameExtension * &bottomExt,
                IFrameExtension * &leftExt,
                IFrameExtension * &rightExt);
```

Searches the frame extensions collection for the last extensions added for the following locations: above, below, left, and right of the client.

flagsFrom

```
static unsigned long flagsFrom(const Style &style);
```

Converts the specified style value to the corresponding system flags.

style Use the styles provided by IFrameWindow::Styles (see page 376) to specify your style.

Motif Information: This function returns the appropriate MWM decoration flags.

Presentation Manager Information: This function returns the corresponding FCF_* flags.

initialize

```
IFrameWindow &initialize(
                const IResourceId &resId,
                const Style &style,
                IWindow *parent = 0,
                IWindow *owner = 0,
                const IRectangle &initRect = nextShellRect ( ),
                const char *title = 0);
```

Initializes the object by creating a standard frame window.

resId The resource identifier of the frame window.

style The default style specification for the frame window. Use the styles provided by IFrameWindow::Styles (see page 376) 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 (see page 361).

title The title of the frame window.

isFrameWindow

Condition: (3) AIX only

```
virtual Boolean isFrameWindow() const;
```

If this is an IFrameWindow object, true is returned.

isRelatedHandle

Condition: (3) AIX only

```
virtual Boolean isRelatedHandle(
                 const IWindowHandle &windowHandle) const;
```

If the specified handle is one of the controls IFrameWindow created during construction, true is returned.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

removeDefaultHandler

```
IFrameWindow &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.

setExtensions

```
IFrameWindow &setExtensions(IFrameExtensions *p);
```

Sets a pointer to the IFrameExtensions (see page 344) collection for the frame window.

p The pointer to the collection of frame extensions.

start

```
IFrameWindow &start(const IWindowHandle &hwnd);
```

Creates a frame handler and adds it to the (newly created) frame window.

hwnd The frame window handle.

styleFrom

```
static unsigned long styleFrom(const Style &style);
```

Converts the style value to the corresponding system flags.

style Use the styles provided by IFrameWindow::Styles (see page 376) to specify your style.

Presentation Manager Information: Converts *style* to the corresponding WS_* flags.

tryToLoadDialog

```
IFrameWindow &tryToLoadDialog(
                const IResourceId &resId,
                IWindow *parent = 0,
                IWindow *owner = 0);
```

Attempts to load the dialog window. If the attempt fails, this function calls IFrameWindow::initialize (see page 371).

resId The resource identifier of the frame window.

parent The parent of the frame window.

owner The owner of the frame window.

Motif Information: AIX does not support this function, therefore it has no effect.

unregisterCallbacks

Condition: (3) AIX only

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

Unregisters the Motif callbacks for the underlying widgets. This function is called during the destruction of the IFrameWindow.

windowSystemClass

```
unsigned long windowSystemClass(const Style &style);
```

This member function is valid only for Microsoft Windows.

Figure 115. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow

Figure 115. Inherited Protected Members

Member function	Class
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Location

```
public:
enum 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.

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.

Motif Information: 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

```
public:
enum 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).

Chapter 82. IFrameWindow::Style

Derivation

IBase
  IBitFlag
    IFrameWindow::Style

Header File: iframe.hpp

The nested class IFrameWindow::Style provides a set of valid frame window styles for the static member functions of the class IFrameWindow (see page 352). You can use these styles in conjunction with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

accelerator
Associates an accelerator key table with the frame window. The frame window constructor loads the table from its resource library.

alignNoAdjust
Restricts adjustment of the frame position. If you specify this style, you lose performance optimizations provided by frame position adjustment.

animated
Shows the frame window with animation when it is opened, closed, or restored.

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.

border
Puts a border around the frame window.

classDefaultStyle
Specifies the original default style for this class, which is titleBar | systemMenu | minimizeButton | windowList | maximizeButton | sizingBorder | appDBCSStatus.

deferCreation
When a derived class creates the frame window, the derived class specifies this style. For example, a derived class could call IFrameWindow::initialize (see page 371), IFrameWindow::create (see page 369), or IFrameWindow::tryToLoadDialog (see page 373) to create a frame window.

© Copyright IBM Corp. 1994

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.

dialogBorder
Puts a dialog border around the frame window.

hideButton
Gives the frame window a hide button.

horizontalScroll
Gives the frame window a horizontal scroll bar.

maximizeButton
Gives the frame window a maximize button.

maximized
Creates the frame window in the maximized state.

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 (see page 499) instead of the menuBar style.

minimizeButton
Gives the frame window a minimize button.

minimized
Creates the frame window in the minimized state.

minimizedIcon
Associates an icon with the frame window. The system uses this icon when it minimizes the frame window.

noMoveWithOwner
Disables the frame window's default behavior of automatically moving with its owner.

shellPosition
Sets the initial position of the window to a location and size dictated by the system shell.

sizingBorder
Puts a sizing border around the frame window.

systemMenu
Gives the frame window a system menu.

systemModal
Sets the frame window in system-modal mode.

titleBar
Gives the frame window a title bar.

verticalScroll
Gives the frame window a vertical scroll bar.

windowList
Adds an entry representing the frame window to the system window list.

Portability Considerations: Some of the supported styles in AIX have different behavior than in OS/2.

Motif Information: AIX does not support the following styles:

alignNoAdjust
animated
appDBCSStatus
dialogBackground
maximized
noMoveWithOwner
shellPosition
windowList

Some of the supported styles have behavior that is slightly different from OS/2 Presentation Manager if you specify:

- sizingBorder, the border is forced on.
- dialogBorder, the frame window has a non-resizing border.
- minimizedIcon, the minimized icon appears in the icon box, but not in the title bar.
- hideButton, the behavior is equivalent to minimizeButton.

Certain styles will affect the system menu function choices. If you do not specify the style:

- minimizeButton, the Minimize function is not available on the system menu.
- maximizeButton, the Maximize function is not available on the system menu.
- sizingBorder, the Size function is not available on the system menu.

Public Members

Figure 116. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 117. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 83. IGraphicPushButton

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              IPushButton
                IGraphicPushButton
```

Header File: igraphbt.hpp

Objects of the IGraphicPushButton class create and manage graphic push button control windows. The standard push button generates an ICommandEvent (see page 128). However, the application can change the window style value to generate a help event or system command event. To change the push button event (that is, message) processing, call IPushButton::enableHelp (see page 699) and IPushButton::enableSystemCommand (see page 700). These functions add or remove the styles IPushButton::help (see page 705) and IPushButton::systemCommand (see page 705), respectively. For example:

```
IGraphicPushButton pbCancel(ID_CANCEL, this, this, ID_SOMEBMP,
                            IRectangle(10,10,80,22));
```

Portability Considerations: AIX does not support dialog templates.

Motif Information: The IGraphicPushButton control consists of a XmPushButton widget. The XmNlabelPixmap resource is set to the graphic specified on the constructor.

Nested Classes

IGraphicPushButton::Style

Constructors

```
public:
IGraphicPushButton(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IResourceId &bitmapOrIconId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```


© Copyright IBM Corp. 1994

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

public:
IGraphicPushButton(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IBitmapHandle &bitmap = IBitmapHandle ( ),
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IGraphicPushButton(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IPointerHandle &icon,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IGraphicPushButton(unsigned long id, IWindow *parentDialog);

public:
IGraphicPushButton(const IWindowHandle &handle);
```

You can construct objects of this class by providing one of the following:

- A nonsystem icon or bit map. Create a graphic push button using the specified window ID, parent and owner windows, screen position and size, resource library with its specified icon or bit-map ID, and window style. If you have not already loaded the bit map or icon and you want to load it from the .dll or .exe of your choice, use this constructor.
- A nonsystem icon or bit map. Create a graphic push button with the specified window ID, parent and owner windows, screen position and size, default resource library with its specified icon or bit-map ID, and window style. If you have not already loaded the bit map or icon and you want to load it from the default resource library (the .exe), use this constructor.
- A nonsystem bit map. Create a graphic push button with the specified window ID, parent and owner windows, screen position and size, bit map, and window style. If you already have a bit-map handle, use this contructor to put the bit map on the graphic push button.
- A nonsystem icon. Create a graphic push button with the specified window ID, parent and owner windows, screen position and size, icon, and window style. If you already have an icon handle, use this constructor to put the icon on a graphic push button.
- A dialog template. Create a graphic push button object for the specified push button that has the icon or bit-map style from the dialog control.

- An existing icon control. Create a graphic push button object for the specified bit-map or icon push button window handle.

Public Members

bitmap

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

Returns the handle of the currently set bit map. If no bit map is set into the graphic push button, an IBitmapHandle (see page 53) is returned.

currentGraphicType

```
GraphicType currentGraphicType() const;
```

Returns the current type of graphic set into the graphic push button. The returned value is an enumerator provided by IGraphicPushButton::GraphicType (see page 387).

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 388) unless you have changed it using setDefaultStyle (see page 383).

disableSizeToGraphic

Condition: (1) Ignored on AIX

```
virtual IGraphicPushButton &disableSizeToGraphic();
```

Removes the style sizeToGraphic (see page 388).

Motif Information: AIX does not support this function and it has no effect.

enableSizeToGraphic

Condition: (1) Ignored on AIX

```
virtual IGraphicPushButton &enableSizeToGraphic(
                            Boolean sizeToGraphic = true);
```

Adds or removes the style sizeToGraphic (see page 388).

Motif Information: AIX does not support this function and it has no effect.

graphicWindow

```
IIconControl &graphicWindow() const;
```

Returns the IIconControl (see page 433) used for drawing the graphic.

Motif Information: AIX does not support this function and it has no effect.

icon

```
IPointerHandle icon() const;
```

Returns the handle of the currently set icon. If no icon is set into the graphic push button, an IIconControl (see page 433) is returned.

isSizeToGraphic

```
Boolean isSizeToGraphic() const;
```

If the style sizeToGraphic (see page 388) is set, true is returned.

marginSize

Condition: (1) Ignored on AIX

```
ISize marginSize() const;
```

Returns the currently set margin size.

moveSizeTo

Condition: (3) AIX only

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

Changes the position and size of the window.

setDefaultStyle

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

Sets the default style for all subsequent graphic push buttons. Use the styles provided by IGraphicPushButton::Style (see page 388) to specify the default style.

setGraphic

```
virtual IGraphicPushButton &setGraphic(
                              const IPointerHandle &handle);

virtual IGraphicPushButton &setGraphic(
                              unsigned long bitmapOrIconId);

virtual IGraphicPushButton &setGraphic(
                              const IResourceId &bitmapOrIconId);

virtual IGraphicPushButton &setGraphic(
                              const IBitmapHandle &handle);
```

Changes the stored graphic to the specified value.

setMarginSize

Condition: (1) Ignored on AIX

```
IGraphicPushButton &setMarginSize(const ISize &size);
```

Changes the size of the margin for the graphic. This only has meaning if the sizeToGraphic style is not set. The height of the ISize will be used for both the top and bottom of the button, and the width value of the ISize will be used for the left and right sides.

Figure 118. Inherited Public Members

Member function	Class
activeColor	IWindow
addBorder	IPushButton
allowsMouseClickFocus	IButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IButton
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
click	IButton
defaultPushButton	IWindow
disable	IWindow
disableBorder	IPushButton
disabledBackgroundColor	IWindow
disableDefault	IPushButton
disabledForegroundColor	IWindow
disabledForegroundColor	IButton
disableGroup	IControl
disableHelp	IPushButton
disableMouseClickFocus	IButton
disableNotification	INotifier
disableNotification	IWindow
disableSystemCommand	IPushButton
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl

Figure 118. Inherited Public Members

Member function	Class
enable	IWindow
enableBorder	IPushButton
enableDefault	IPushButton
enableGroup	IControl
enableHelp	IPushButton
enableMouseClickFocus	IButton
enableNotification	INotifier
enableNotification	IWindow
enableSystemCommand	IPushButton
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IButton
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
highlight	IButton
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IButton
hiliteForegroundColor	IWindow
hiliteForegroundColor	IButton
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isBorder	IPushButton
isDefault	IPushButton
isDisabled	IWindow

Figure 118. Inherited Public Members

Member function	Class
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHelp	IPushButton
isHighlighted	IButton
isLayoutDistorted	IWindow
isShowing	IWindow
isSystemCommand	IPushButton
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeBorder	IPushButton
resetActiveColor	IWindow
resetBackgroundColor	IWindow

Figure 118. Inherited Public Members

Member function	Class
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IButton
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl

Figure 118. Inherited Public Members

Member function	Class
topHandle	IWindow
unhighlight	IButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the recommended minimum size of this graphic push button control. The size is based on the actual size of the stored graphic.

Figure 119. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
calcMinimumSize	IPushButton
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	IPushButton
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IPushButton
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier

Figure 119. Inherited Protected Members

Member function	Class
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IPushButton
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

GraphicType

```
public:
enum GraphicType
  {
  bitmapType,
  iconType,
  };
```

These enumerators specify the current type of graphic set into the graphic push button:

bitmapType
Specifies a bit map is set into the graphic push button. If you do not specify a handle or resource ID when constructing the graphic push button object, an IBitmapHandle (see page 53) is the default graphic. This is the default style.

iconType
Specifies an icon is set into the graphic push button.

Chapter 84. IGraphicPushButton::Style

Derivation

IBase
  IBitFlag
    IGraphicPushButton::Style

Header File: igraphbt.hpp

The nested class IGraphicPushButton::Style provides a set of valid graphic button styles for the static member functions of the class IGraphicPushButton (see page 380). You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IButton::Style (see page 61)
IPushButton::Style (see page 705)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

sizeToGraphic
Displays the graphic inside of the push button using the graphic's actual size. If the graphic is larger than the push button, the graphic is clipped. If the graphic is smaller than the push button, the graphic is centered in the push button. If you do not specify this style, the graphic is stretched to fill the push button within its border completely.

Public Members

Figure 120. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 121. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 85. IGroupBox

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IGroupBox
```

Header File: igroupbx.hpp

Objects of the IGroupBox class create and manage the 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 accepts input.

Motif Information: The AIX release of the User Interface Class Library does not support the IGroupBox class objects.

Nested Classes

IGroupBox::Style

Constructors

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

public:
IGroupBox(unsigned long id, IWindow *parentDialog);

public:
IGroupBox(const IWindowHandle &handle);
```

You can construct objects of this class by using the following:

- Parent window, owner window, optional size and location, and optional style parameters

© Copyright IBM Corp. 1994

- Parent window
- Handle of an existing group box

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. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0.

owner Owner window of the group box control you construct.

initial (Optional) The initial position and size of the group box control you construct.

style (Optional) Group box control's characteristics.

parentDialog
The parent window.

handle The window handle of an existing group box control.

Public Members

defaultStyle

Condition: (1) Ignored on AIX

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

Returns the default style. The default style is classDefaultStyle (see page 396) unless you have changed it using setDefaultStyle (see page 392).

foregroundColor

Condition: (1) Ignored on AIX

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

Returns the foreground color value of the window area, or the default if no color for the area has been set.

moveSizeTo

Condition: (3) AIX only

Condition: (1) Ignored on AIX

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

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.

position

Condition: (3) AIX only

Condition: (1) Ignored on AIX

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

Returns the window's position converted to the coordinates understood by this application from window system coordinates (if they are different).

rect

Condition: (3) AIX only

Condition: (1) Ignored on AIX

```
virtual IRectangle rect() const;
```

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.

setDefaultStyle

Condition: (1) Ignored on AIX

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

Sets the default style for all subsequent group boxes.

style Use the styles provided by IGroupBox::Styles (see page 396) to specify the default style.

setText

Condition: (1) Ignored on AIX

```
virtual IGroupBox &setText(const char *text);
```

Condition: (1) Ignored on AIX

```
virtual IGroupBox &setText(const IResourceId &text);
```

Sets the text for the group box. If children exist for an parent IMultiCellCanvas, the parent canvas is notified to update the layout for its children.

size

Condition: (3) AIX only

Condition: (1) Ignored on AIX

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

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.

Figure 122. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow

Figure 122. Inherited Public Members

Member function	Class
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow

Figure 122. Inherited Public Members

Member function	Class
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow

Figure 122. Inherited Public Members

Member function	Class
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

Condition: (1) Ignored on AIX

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

Returns the minimum size of the group box control based on the current font and the text string.

Figure 123. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow

Figure 123. Inherited Protected Members

Member function	Class
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 86. IGroupBox::Style

Derivation

IBase
  IBitFlag
    IGroupBox::Style

Header File: igroupbx.hpp

The nested class IGroupBox::Style provides a set of valid group box styles for the member functions of the class IGroupBox (see page 390). You can use this style with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

Public Members

Figure 124. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 125. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 87. IGUIColor

Derivation

IBase
  IVBase
    IColor
      IGUIColor

Header File: icolor.hpp

You can use IGUIColor objects to query the color of a system area.

Portability Considerations: In AIX, this class only provides hard-coded, default color values. If you want your application to be portable, use the class IColor (see page 103) to create IColor objects. Such objects give you control over color values.

Motif Information: 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.

Presentation Manager Information: The IGUIColor class's objects correspond to one of the system SYSCLR_* colors in Presentation Manager.

Constructors

```
public:
IGUIColor(SysColor value);
```

You can only construct objects of this class from a system color value. The constructor sets the internal system color value using the specified SysColor (see page 398) enumerator.

Public Members

sysColor

```
SysColor sysColor() const;
```

Returns the SysColor enumerator used to create this object.

© Copyright IBM Corp. 1994

Figure 126. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asPixel	IColor
asRGBLong	IColor
asString	IBase*
asString	IVBase*
blueMix	IColor
greenMix	IColor
index	IColor
operator !=	IColor
operator <<	IBase*
operator <<	IVBase*
operator =	IColor
operator ==	IColor
redMix	IColor
setBlue	IColor
setGreen	IColor
setRed	IColor
value	IColor

Note: * Application Support Class Library

Enumerations

SysColor

```
public:
enum SysColor
  {
  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 the preceding enumerators to specify system color values.

Chapter 88. IHandle

Derivation

IBase
  IHandle

Header File: ihandle.hpp

The IHandle class is a base class for all 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 storing 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.

Motif Information: Typically, this value originates in an X API call.

Presentation Manager Information: Typically, this value originates in an OS/2 system API call.

Constructors

```
public:
IHandle(Value value);
```

You can construct objects of this by providing an IHandle type value.

Public Members

asDebugInfo

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


© Copyright IBM Corp. 1994

Returns the handle as a string containing diagnostic information.

asString

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

Returns the handle as a string of form `nnnn`.

Figure 127. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 89. IHandler

Derivation

IBase
  IVBase
    IHandler

Header File: ihandler.hpp

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 IWindow::addHandler (see page 944). Windows can also add handlers by using IHandler::handleEventsFor (see page 403).

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 (see page 404).

The handler that processes a given event is also responsible for setting the appropriate event result. The handler does this by using IEvent::setResult (see page 281). 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 IWindow::removeHandler (see page 949) or IHandler::stopHandlingEventsFor (see page 403). 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.

Motif Information: If none of the handlers returns true, nothing happens.

Presentation Manager Information: If none of the handlers returns true, the original window procedure for the underlying Presentation Manager window is called.

Constructors

```
public:
IHandler();
```

Because this is an abstract base class, you can only call the constructor from the constructors of derived classes. The constructor accepts no parameters and initializes the handler in an enabled state.

© Copyright IBM Corp. 1994

Public Members

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

Returns diagnostic information about the handler.

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

Returns the string "IHandler(enabled)" or "IHandler(disabled)."

disable
```
virtual IHandler &disable();
```

Disables the handler so that it processes no window events.

enable
```
virtual IHandler &enable(Boolean enable = true);
```

Enables the handler, by default. It accepts an optional *Boolean* parameter that, when false, causes the handler to be disabled.

handleEventsFor
```
virtual IHandler &handleEventsFor(IWindow *window);
```

Attaches the handler to the specified IWindow (see page 924) object.

isEnabled
```
Boolean isEnabled() const;
```

Returns whether the handler is currently enabled.

stopHandlingEventsFor
```
virtual IHandler &stopHandlingEventsFor(IWindow *window);
```

Detaches the handler from the specified IWindow (see page 924) object.

Figure 128. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*

Figure 128. Inherited Public Members

Member function	Class
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

defaultProcedure

```
virtual IEventResult defaultProcedure(IEvent &event);
```

Sends the specified IEvent (see page 275) object to the original window procedure for the underlying window.

Motif Information: AIX does not support this function.

dispatchHandlerEvent

```
virtual Boolean dispatchHandlerEvent(IEvent &event) = 0;
```

IWindow (see page 924) 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 (see page 275) object.

Chapter 90. IHelpErrorEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpErrorEvent

Header File: ihelpevt.hpp

Objects of the IHelpErrorEvent class are dispatched when user interaction causes a help error.

Constructors

```
public:
IHelpErrorEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

error

```
ErrorType error() const;
```

Returns the type of error caused by user interaction. The returned value is an ErrorType (see page 406) enumerator.

Figure 129. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent

Figure 129. Inherited Public Members

Member function	Class
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent

© Copyright IBM Corp. 1994

Figure 129. Inherited Public Members

Member function	Class
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Enumerations

ErrorType

```
public:
enum ErrorType
  {
  loadingDLL,
  noWindow,
  invalidAppHandle,
  invalidInstanceHandle,
  noHelpInstance,
  invalidQueryHandle,
  noHelpTable,
  invalidId,
  noIndex,
  noContent,
  openHelpFile,
  readHelpFile,
  closeHelpFile,
  invalidHelpFile,
  notEnoughMemory,
  unableFreeMemory,
  invalidHelpWindow,
  unopenDatabase,
  fontNotFound,
  unknown,
  };
```

These enumerators specify the type of error that occurred:

loadingDLL
Specifies the application is unable to load the resource dynamic link library (DLL).

noWindow
Specifies there is no frame window in the window chain from which to find or set the associated help instance.

invalidAppHandle
Specifies the application window handle to be associated with the help object is not a valid window handle.

invalidInstanceHandle
Specifies the help object handle to be associated with the application window is not a valid window handle.

noHelpInstance
Specifies the parent or owner chain of the specified application window does not have an associated help object.

invalidQueryHandle
Specifies the application window handle used to query the help object is not a valid window handle.

noHelpTable
Specifies the application did not provide a help table for context-sensitive help.

invalidId Specifies the ID of the main help item specified for a context-sensitive help request was not found in the help table.

noIndex Specifies the index is not in the library file.

noContent Specifies the library file does not have any content.

openHelpFile
Specifies the library file cannot be opened.

readHelpFile
Specifies the library file cannot be read.

closeHelpFile
Specifies the library file cannot be closed.

invalidHelpFile
Specifies an improper library file was provided.

notEnoughMemory
Specifies unable to allocate the requested amount of memory.

unableFreeMemory
Specifies unable to free allocated memory.

invalidHelpWindow
Specifies unable to find the requested help window.

unopenDatabase
Specifies unable to read the unopened database.

fontNotFound
Specifies the font could not be loaded.

unknown Specifies another error.

Chapter 91. IHelpHandler

Derivation

IBase
  IVBase
    IHandler
      IHelpHandler

Header File: ihelphdr.hpp

Objects of the IHelpHandler class process help window events.

Create a handler derived from IHelpHandler and attach it to the application frame window whose help events you want to handle. You can do this by calling IHelpHandler::handleEventsFor (see page 409) to pass the appropriate window to the help handler.

IHelpHandler processes the following help events:

- Notification of any of the following:
  - Application-added control selected on a help window
  - Library opened
  - Table of contents shown
  - Cover page shown
  - History window shown
  - Index shown
  - Page shown
  - Search list shown
  - Page swapped

  This is described in IHelpNotifyEvent (see page 416).
- Errors caused by user interaction. This is described in IHelpErrorEvent (see page 405).
- Requests for help when either:
  - General help is undefined
  - The user selects **Keys Help**

  This is described in IEvent (see page 275).
- Reference to a heading containing a link that requests that the application be informed. This is described in IHelpHypertextEvent (see page 413).
- Selection of an application-added item on a customized help menu bar. This is described in IHelpMenuBarEvent (see page 415).
- Selection of the Tutorial choice from the Help menu. This is described in IHelpTutorialEvent (see page 420).
- Requests for help on a field for which help is not found. This is described in IHelpSubitemNotFoundEvent (see page 418).

© Copyright IBM Corp. 1994

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 help event to the next handler.
- If this is the last handler for the application frame window, call IWindow::defaultProcedure (see page 946) to process the help event.

Public Members

handleEventsFor

```
virtual IHelpHandler &handleEventsFor(
                        IFrameWindow *associatedWindow);
```

Attaches the handler to a frame window. The application window must be one of the following:

- The window you specified in constructing the IHelpWindow (the primary window)
- The secondary window using the primary window's help
- The secondary window using its own help

stopHandlingEventsFor

```
virtual IHelpHandler &stopHandlingEventsFor(
                        IFrameWindow *associatedWindow);
```

Detaches the handler from the application window.

Figure 130. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler

Figure 130. Inherited Public Members

Member function	Class
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

controlSelect
```
virtual Boolean controlSelect(IHelpNotifyEvent &event);
```

Called when the user selects an application-added control on a help window. By default, this function does nothing.

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

If a help event is received, the appropriate virtual function is called.

handleError
```
virtual Boolean handleError(IHelpErrorEvent &event);
```

Called when a user interaction causes an error. By default, this function displays a message box indicating the error type of the IHelpErrorEvent (see page 405). The error type is an enumerator provided by ErrorType (see page 406).

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

Called when the user requests general help and it is not defined. By default, this function does nothing. Your application can override this function and either:

- Display its own window
- Use IHelpWindow::show (see page 426) to display a particular window.

hypertextSelect
```
virtual Boolean hypertextSelect(IHelpHypertextEvent &event);
```

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.

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

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 false, which indicates that no action is required from the Information Presentation Facility (IPF). You can override the default action and set the result to the ID of the help window for IPF to display.

menuBarCommand

```
virtual Boolean menuBarCommand(IHelpMenuBarEvent &event);
```

Called when the user selects an application-added item on a customized help menu bar. By default, this function does nothing.

openLibrary

Condition: (1) Ignored on AIX

```
virtual Boolean openLibrary(IHelpNotifyEvent &event);
```

Called when a new library is opened. By default, this function does nothing.

showContents

Condition: (1) Ignored on AIX

```
virtual Boolean showContents(IHelpNotifyEvent &event);
```

Called when the table of contents window is displayed. By default, this function does nothing.

showCoverPage

Condition: (1) Ignored on AIX

```
virtual Boolean showCoverPage(IHelpNotifyEvent &event);
```

Called when the cover page is displayed. By default, this function does nothing.

showHistory

Condition: (1) Ignored on AIX

```
virtual Boolean showHistory(IHelpNotifyEvent &event);
```

Called when the history window is displayed. By default, this function does nothing.

showIndex

Condition: (1) Ignored on AIX

```
virtual Boolean showIndex(IHelpNotifyEvent &event);
```

Called when the index window is displayed. By default, this function does nothing.

showPage

Condition: (1) Ignored on AIX

```
virtual Boolean showPage(IHelpNotifyEvent &event);
```

Called when a child window of the cover page is opened. By default, this function does nothing.

showSearchList

Condition: (1) Ignored on AIX

```
virtual Boolean showSearchList(IHelpNotifyEvent &event);
```

Called when the search list is displayed. By default, this function does nothing.

showTutorial

```
virtual Boolean showTutorial(IHelpTutorialEvent &event);
```

Called when the user selects the **Tutorial** choice from a help window or from a menu. By default, this function does nothing.

subitemNotFound

```
virtual Boolean subitemNotFound(
                  IHelpSubitemNotFoundEvent &event);
```

Called when the user requests help on a field that the Information Presentation Facility (IPF) cannot find in the help subtable. By default, this function sets the event result to false. IPF then displays the general help window.

swapPage

Condition: (1) Ignored on AIX

```
virtual Boolean swapPage(IHelpNotifyEvent &event);
```

Called when a child window of the cover page is swapped. By default, this function does nothing.

Figure 131. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 92. IHelpHypertextEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpHypertextEvent

Header File: ihelpevt.hpp

Objects of the IHelpHypertextEvent class are only 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.
```

Constructors

```
public:
IHelpHypertextEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

id

```
unsigned long id() const;
```

Returns the resource ID of the inform link. Your application uses this ID to determine application-specific processing for the link.

© Copyright IBM Corp. 1994

Figure 132. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 93. IHelpMenuBarEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpMenuBarEvent

Header File: ihelpevt.hpp

Objects of the IHelpMenuBarEvent class are dispatched when the user selects an application-added item on a customized help menu bar.

Constructors

```
public:
IHelpMenuBarEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

commandId

```
unsigned long commandId() const;
```

Returns the ID of the menu bar item selected.

Figure 133. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*

Figure 133. Inherited Public Members

Member function	Class
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 94. IHelpNotifyEvent

Derivation

```
IBase
  IVBase
    IEvent
      IHelpNotifyEvent
```

Header File: ihelpevt.hpp

Objects of the IHelpNotifyEvent class are dispatched when the user initiates an event that your associated application window might be interested in controlling. Your application can then change the behavior or appearance of the help window.

An IHelpNotifyEvent represents one of the following types:

- Application-added control selected on a help window
- Library opened
- Table of contents shown
- Cover page shown
- History window shown
- Index shown
- Page shown
- Search list shown
- Page swapped

The IHelpNotifyEvent object identifies the event type, the help control that the user selected, and the help window containing the selected control.

Constructors

```
public:
IHelpNotifyEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

controlId

```
unsigned long controlId() const;
```

Returns the resource ID of the help control that the user selected. For example, this might be the resource ID of an application-added push button on a help window.

© Copyright IBM Corp. 1994

windowHandle

```
IWindowHandle windowHandle() const;
```

Returns the window handle of the help window from which the help event was generated. This help window contains the help control that the user selected.

Figure 134. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 95. IHelpSubitemNotFoundEvent

Derivation

```
IBase
  IVBase
    IEvent
      IHelpSubitemNotFoundEvent
```

Header File: ihelpevt.hpp

Objects of the IHelpSubitemNotFoundEvent class are dispatched when the user requests help on a field for which the system cannot find a related entry in the help subtable.

Constructors

```
public:
IHelpSubitemNotFoundEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

isFrame

```
Boolean isFrame() const;
```

Queries whether help was requested on a frame window.

isMenu

```
Boolean isMenu() const;
```

Queries whether help was requested on a menu window.

isWindow

```
Boolean isWindow() const;
```

Queries whether help was requested on an application window.

subtopicId

```
unsigned long subtopicId() const;
```


© Copyright IBM Corp. 1994

If help was requested on an application or frame window, the control ID of the field containing the cursor is returned. If help was requested for a menu item, its ID is returned.

topicId

```
unsigned long topicId() const;
```

Returns the ID of the window or menu that contains the field upon which help was requested.

Figure 135. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 96. IHelpTutorialEvent

Derivation

IBase
  IVBase
    IEvent
      IHelpTutorialEvent

Header File: ihelpevt.hpp

Objects of the IHelpTutorialEvent class are dispatched when the user selects the Tutorial choice from the Help menu.

Constructors

```
public:
IHelpTutorialEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IHelpHandler::dispatchHandlerEvent (see page 410) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

tutorialName

```
IString tutorialName() const;
```

Returns the default tutorial program name. You specify this program when creating the IHelpWindow (see page 422).

Figure 136. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent

Figure 136. Inherited Public Members

Member function	Class
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent

© Copyright IBM Corp. 1994

Figure 136. Inherited Public Members

Member function	Class
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 97. IHelpWindow

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IHelpWindow
```

Header File: ihelp.hpp

Objects of the IHelpWindow class provide help for application windows using the Information Presentation Facility (IPF). Typically, you can create and associate an IHelpWindow object with one of the application's main windows. Multiple help windows can 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.

Motif Information: You can build the viewable help files for your AIX application using the Information Presentation Facility compiler (IPFC) that comes with the OS/2 2.1 Toolkit.

The Motif implementation of IHelpWindow uses the Information Presentation Facility/X (IPF/X) to format and present help text. The IHelpWindow implementation adds support of OS/2-style help tables to the basic IPF/X functionality. Help tables let you set up a mapping between the window IDs used in your application and the resource IDs used in your IPF help text. You can use the ipmrc2X utility provided with the User Interface Class Library to convert the help tables in an existing Presentation Manager application to the format used in Motif.

If you do not provide a help table, IHelpWindow uses the window ID of the IWindow object as the ID of the contextual or general help panel to display.

Nested Classes

IHelpWindow::Settings

Constructors

```
public:
IHelpWindow(IFrameWindow *associatedWindow = 0);

public:
IHelpWindow(
   const IResourceId &helpTable,
   IFrameWindow *associatedWindow);
```


© Copyright IBM Corp. 1994

```
public:
IHelpWindow(
   const Settings &settings,
   IFrameWindow *associatedWindow = 0);

public:
IHelpWindow(const IWindowHandle &helpWindowHandle);
```

You can construct objects of this class in the following ways:

- By providing an optional frame window which:
  - The help window references to position itself
  - Receives help notification events
- By providing a resource ID to identify the help table in addition to a frame window which:
  - The help window references to position itself
  - Receives help notification events
- By providing an object of the class IHelpWindow::Settings (see page 431) to associate the help with initial information such as a:
  - Modified menu bar
  - Accelerator table
  - Title
  - Tutorial
  - Library names

  The application can also use *associatedWindow* to specify a frame window which:
  - The help window references to position itself
  - Receives help notification events

  This associates the application with the help object when the application calls the constructor. Otherwise, the application can call IHelpWindow::associateWindow (see page 424) after construction.
- From an existing help window by specifying the window handle.

Public Members

addLibraries

```
virtual IHelpWindow &addLibraries(
                        const char *helpLibraryNames);
```

Adds a library or list of libraries to those referenced by the Information Presentation Facility (IPF) when looking for help windows by their ID. If you specify multiple library names, separate each name with a blank space. If you specify a null string for *helpLibraryNames*, IPF no longer references any library names that were previously specified using this function. This member function deals with binary help files that have an .hlp extension.

associateWindow

```
IHelpWindow &associateWindow(IFrameWindow *frameWindow);
```

Sets the application window chain to receive help events. More than one window can be associated with a help window.

You can also use this function to manage help windows for a secondary application window. This might be necessary for the following reasons:

- To position the help windows relative to the secondary application window
- To close the help windows when the secondary application window closes

Note: This function is obsolete. Use IHelpWindow::setAssociatedWindow instead.

communicationWindow

```
IWindowHandle communicationWindow() const;
```

Returns the handle of the active communication window.

contentsWindow

```
IWindowHandle contentsWindow() const;
```

Returns the handle of the Table of Contents window.

coverPageWindow

```
IWindowHandle coverPageWindow() const;
```

Returns the handle of the Information Presentation Facility (IPF) multiple document interface parent window.

handle

```
virtual IWindowHandle handle() const;
```

Returns the help object window handle.

helpForWindow

```
IFrameWindow *helpForWindow() const;
```

Returns the current active window. If there is not an active window, 0 is returned.

Note: This function is obsolete; there is no replacement function.

helpWindow

```
static IHelpWindow *helpWindow(const IWindow *window);
```

Returns the help window that is associated with the specified application window. The parent window chain is searched until an application window associated with the help window is found. If a help window cannot be found, 0 is returned.

hide

```
virtual IHelpWindow &hide();
```

Closes the help window.

hidePanelIds

```
virtual IHelpWindow &hidePanelIds();
```

Removes the help window's panel ID from the help window's title.

indexWindow

```
IWindowHandle indexWindow() const;
```

Returns the handle of the Index window.

moveSizeTo

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

Changes the size and position of the help cover page window (the window within which all other Information Presentation Facility (IPF) windows are displayed).

moveTo

```
virtual IHelpWindow &moveTo(const IPoint &aPoint);
```

Changes the position of the help cover page window, within which all other Information Presentation Facility (IPF) windows are displayed.

searchListWindow

```
IWindowHandle searchListWindow() const;
```

Returns the handle of the Search window.

sendEvent

```
virtual IEventResult sendEvent(const IEvent &event) const;

virtual IEventResult sendEvent(
                        unsigned long eventId,
                        const IEventParameter1 &parm1 = 0,
                        const IEventParameter2 &parm2 = 0) const;

virtual IEventResult sendEvent(
                        EventType eventType,
                        const IEventParameter1 &parm1 = 0,
                        const IEventParameter2 &parm2 = 0) const;
```

Overridden to ensure that events get routed to the correct handler.

setActiveWindow

```
virtual IHelpWindow &setActiveWindow(
                         IFrameWindow *activeWindow,
                         IFrameWindow *relativeWindow = 0);
```

Sets the application window to receive help window events. Information Presentation Facility (IPF) considers this to be the window that help is requested for. You can enable contextual and general help for a child window in your application using this function.

Optionally, you can specify the window to position the help window next to.

Help events are sent to the window specified by setAssociatedWindow until another window is made the active window by calling setActiveWindow.

setAssociatedWindow

```
virtual IHelpWindow &setAssociatedWindow(
                      IFrameWindow *associatedWindow);
```

Sets the application window chain to receive help events. More than one window can be associated with a help window.

You can also use this function to manage help windows for a secondary application window. This might be necessary for the following reasons:

- To position the help windows relative to the secondary application window
- To close the help windows when the secondary application window closes

setHelpTable

```
virtual IHelpWindow &setHelpTable(
                      const IResourceId &helpTable);
```

Sets the help table used to identify help windows.

setTitle

```
virtual IHelpWindow &setTitle(const IResourceId &titleId);

virtual IHelpWindow &setTitle(const char *titleText);
```

Sets the title text for the help window. You can specify the text either as a text string or as a resource ID to load.

setUsingHelp

```
virtual IHelpWindow &setUsingHelp(unsigned long panelId);
```

Sets the application help panel to use instead of the default Using Help window.

show

```
virtual IHelpWindow &show(const IResourceId &panelId);

virtual IHelpWindow &show(HelpType helpType);

virtual IHelpWindow &show(const char *panelName);
```

Displays a help window. You can specify the help window using any of the following:

- The help panel name.

- An enumerator provided by HelpType (see page 430).
- The help window's ID, which the library uses to load the help window. This is the resource ID you specify in the IPF file. If you specify a window ID of 0, the Using Help window is displayed.

showPanelIds

```
virtual IHelpWindow &showPanelIds(Boolean visibleId = true);
```

Adds the help window's panel ID to the help window's title.

sizeTo

```
virtual IHelpWindow &sizeTo(const ISize &aSize);
```

Changes the size of the help cover page window, within which all other Information Presentation Facility (IPF) windows are displayed.

viewedPagesWindow

```
IWindowHandle viewedPagesWindow() const;
```

Returns the handle of the Viewed Pages window.

Figure 137. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableUpdate	IWindow

Figure 137. Inherited Public Members

Member function	Class
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow

Figure 137. Inherited Public Members

Member function	Class
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow

Figure 137. Inherited Public Members

Member function	Class
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 138. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

HelpType

```
public:
enum HelpType
  {
  index,
  general,
  contents,
  keys,
  using,
  };
```

Use these enumerators to specify the type of help window to display:

index The Help Index window.

general The General Help window.

contents The Help Contents window.

keys The Keys Help window.

using The Using Help window.

Chapter 98. IHelpWindow::Settings

Derivation

IBase
  IHelpWindow::Settings

Header File: ihelp.hpp

Objects of the IHelpWindow::Settings class provide initial information for the constructor of the IHelpWindow class. Typically, you can create a Settings object as well as set some or all of the data before passing the object to the IHelpWindow (see page 422) constructor that accepts a Settings object.

Constructors

```
public:
Settings();
```

You can only construct objects of this class using the default constructor which accepts no parameters.

Public Members

setAccelerator

```
Settings &setAccelerator(unsigned long acceleratorId);
```

If a customized help menu bar is used for your help window, this member function specifies an accelerator table to use with it.

Motif Information: This function has no effect in AIX.

setAccelResLibrary

```
Settings &setAccelResLibrary(
            const char *menuAccelResLibrary);
```

Sets the resource dynamic link library (DLL) that contains the help window menu accelerator table. You must include the .dll extension as part of the specified string. The default setting is the executable file.

Motif Information: This function has no effect in AIX.

setHelpResLibrary

```
Settings &setHelpResLibrary(
            const char *helpTableResLibrary);
```

© Copyright IBM Corp. 1994

Sets the resource DLL that contains the help table. The default setting is the executable file.

Motif Information: This function has no effect in AIX.

setHelpTable

```
Settings &setHelpTable(unsigned long helpTable);
```

Sets the help table used to identify help windows.

setLibraries

```
Settings &setLibraries(const char *helpLibraryNames);
```

Sets a library or list of libraries for the Information Presentation Facility (IPF) to reference when looking for help windows by their ID. If you specify multiple library names, separate the names with a blank space. This member function deals with binary help files that have an .hlp extension.

setMenuBar

```
Settings &setMenuBar(unsigned long menuBarId);
```

Sets the menu bar template to use instead of the Information Presentation Facility (IPF) default menu bar.

setTitle

```
Settings &setTitle(const IResourceId &titleId);

Settings &setTitle(const char *titleText);
```

Sets the text to use in the title of the help window. You can specify the text either as a text string or as a resource ID to load.

setTutorial

```
Settings &setTutorial(const char *tutorial);
```

Sets the name of the application's tutorial. If you set a tutorial name, the Information Presentation Facility (IPF) provides a **Tutorial** choice on the menu bar of the help window.

setUsingHelp

```
Settings &setUsingHelp(unsigned long panelId);
```

Sets the application help panel to use instead of the default Information Presentation Facility (IPF) Using Help window.

Figure 139. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*

Figure 139. Inherited Public Members

Member function	Class
operator <<	IBase*

Note: * Application Support Class Library

Chapter 99. IIconControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IStaticText
              IBitmapControl
                IIconControl

Header File: iiconctl.hpp

Objects of the IIconControl class create and operate upon icon controls. You can also use this class to add text to an icon by calling ITextControl::setText (see page 875).

Motif Information: AIX does not support system bit maps and dialog templates.

Nested Classes

IIconControl::Style

Constructors

```
public:
IIconControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IResourceId &iconId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IIconControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   unsigned long iconId,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IIconControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   const IPointerHandle &iconId = IPointerHandle ( ),
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

© Copyright IBM Corp. 1994

Condition: (2) OS/2 Only

```
public:
IIconControl(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
   ISystemPointerHandle::Identifier icon,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IIconControl(unsigned long id, IWindow *parentDialog);

public:
IIconControl(const IWindowHandle &handle);
```

You can construct objects of this class from the following:

- A non-system icon. This creates an icon control with the specified window ID, parent and owner windows, screen position and size, icon and resource library ID, and window style.
- A non-system icon. This creates an icon control with the specified window ID, parent and owner windows, screen position and size, icon ID and window style.
- A non-system icon. This creates an icon control with the specified window ID, parent and owner windows, screen position and size, icon pointer handle and window style.
- A system icon. This creates an icon control with the specified window ID, parent and owner windows, screen position and size, system icon, and window style.
- A dialog template. This creates an IIconControl object for the specified icon dialog control.
- An existing icon control. This creates an IIconControl object for the specified icon control's window handle.

Public Members

defaultStyle

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

Returns the default style. This is classDefaultStyle (see page 439) unless you have changed it using setDefaultStyle (see page 435).

icon

```
IPointerHandle icon() const;
```

Returns the handle of the currently set icon.

moveSizeTo

Condition: (3) AIX only

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

Changes the position and size of the window.

setDefaultStyle

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

Sets the default style for all subsequent icon controls.

setIcon

Condition: (2) OS/2 Only

```
virtual IIconControl &setIcon(
                        ISystemPointerHandle::Identifier icon);

virtual IIconControl &setIcon(unsigned long iconId);

virtual IIconControl &setIcon(const IResourceId &iconId);

virtual IIconControl &setIcon(const IPointerHandle &handle);
```

Sets the icon handle of the icon control window.

Figure 140. Inherited Public Members

Member function	Class
activeColor	IWindow
alignment	IStaticText
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IStaticText
bitmap	IBitmapControl
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableFillBackground	IStaticText

Figure 140. Inherited Public Members

Member function	Class
disableGroup	IControl
disableHalftone	IStaticText
disableNotification	INotifier
disableNotification	IWindow
disableStrikeout	IStaticText
disableTabStop	IControl
disableUnderscore	IStaticText
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableFillBackground	IStaticText
enableGroup	IControl
enableHalftone	IStaticText
enableNotification	INotifier
enableNotification	IWindow
enableStrikeout	IStaticText
enableTabStop	IControl
enableUnderscore	IStaticText
enableUpdate	IWindow

Figure 140. Inherited Public Members

Member function	Class
fillColor	IStaticText
font	IWindow
foregroundColor	IWindow
foregroundColor	IStaticText
handle	IWindow
handleException	IWindow
hasFillBackground	IStaticText
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHalftone	IStaticText
isLayoutDistorted	IWindow
isShowing	IWindow
isStrikeout	IStaticText
isTabStop	IWindow
isTabStop	IControl
isUnderscore	IStaticText
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
limit	IStaticText
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveSizeTo	IStaticText

Figure 140. Inherited Public Members

Member function	Class
moveSizeTo	IBitmapControl
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetFillColor	IStaticText
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IStaticText
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBitmap	IBitmapControl
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFillColor	IStaticText

Figure 140. Inherited Public Members

Member function	Class
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setLimit	IStaticText
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IStaticText
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size this icon control can be based on the actual size of the currently set icon.

Figure 141. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier

Figure 141. Inherited Protected Members

Member function	Class
addObserver	IWindow
calcLimitSize	IStaticText

Figure 141. Inherited Protected Members

Member function	Class
calcMinimumSize	IWindow
calcMinimumSize	IStaticText
calcMinimumSize	IBitmapControl
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	IStaticText
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 100. IIconControl::Style

Derivation

IBase
  IBitFlag
    IIconControl::Style

Header File: iiconctl.hpp

The nested class IIconControl::Style provides a set of valid icon control styles for the static member functions of the class IIconControl (see page 433). You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IStatic::Style (see page 846)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

sizeToIcon
Sizes the window to the size of the icon specified on the constructor.

Public Members

Figure 142. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

© Copyright IBM Corp. 1994

Figure 143. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 101. IInfoArea

Derivation

```
IBase                IBase
  IVBase               IVBase
    INotifier            INotifier
      IWindow              IHandler
        IControl           IMenuHandler
          ITextControl         IInfoArea
            IStaticText
              IInfoArea
```

Header File: iinfoa.hpp

Objects of the IInfoArea class provide a frame extension at the bottom of the client area. The frame extension shows information about the frame menu item at which the selection cursor is currently positioned.

The information area displays the contents of an entry in a resource string table. The string resource is obtained by using the same ID as the menu item plus some optional offset value that is added to the menu item ID to locate the corresponding string of information about it.

Motif Information: The User Interface Class Library loads the strings from the user default library.

Unlike OS/2 Presentation Manager, Motif does not update the information area for menu bar items (CascadeButtons). Only pull-down menu item (ToggleButton) information is presented by the information area.

Presentation Manager Information: The User Interface Class Library loads the strings from a dynamic link library (.dll). You can specify a library name, or you can use 0 to indicate that the the default user application resource library supplies the resources.

Constructors

```
public:
IInfoArea(IFrameWindow *frame, unsigned long id = 0);
```

```
public:
IInfoArea(
   IFrameWindow *frame,
   unsigned long id,
   const char *resDLLName);
```

© Copyright IBM Corp. 1994

```
public:
IInfoArea(
   IFrameWindow *frame,
   const IModuleHandle &resMod,
   unsigned long id = 0);

public:
IInfoArea(
   IFrameWindow *frame,
   const char *resDLLName,
   unsigned long id = 0);
```

You can construct objects of this class with the following required information:

- A pointer to the frame window you want to attach the information area to.
- The ID you specify for the information area control. The default is 0.
- The resource library from which you want to load the information strings. You can specify this library using any of the following:
  - The module handle of the resource .dll
  - A module handle of 0, which indicates the current .exe
  - The name of a .dll file
  - The name of a library

  The default is the user default library obtained from userResourceLibrary (see page 233).

The second and third parameters are optional. You can specify either or both, in any order.

Public Members

disabledText

```
virtual IString disabledText() const;
```

Returns the text displayed when the menu item at the selection cursor is disabled.

handleEventsFor

```
virtual IInfoArea &handleEventsFor(IFrameWindow *frame);
```

Attaches the handler of the information area to the specified frame window.

inactiveText

```
virtual IString inactiveText() const;
```

Returns the text displayed when the menu is inactive.

lineCount

Condition: (2) OS/2 Only

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

Returns the number of text lines that the information area uses.

missingText

```
virtual IString missingText() const;
```

Returns the text that displays when the User Interface Class Library cannot obtain the required information string from the resource library.

resourceLibrary

```
virtual IResourceLibrary &resourceLibrary() const;
```

Returns a reference to the library being used.

setDisabledText

```
virtual IInfoArea &setDisabledText(
                      unsigned long disabledText);

virtual IInfoArea &setDisabledText(
                      const IString &disabledText);
```

Sets the text that is displayed whenever a disabled menu item becomes the current menu item. To display the text, you must call this function before the disabled menu item is made the current menu item. Otherwise, the normal information area text is displayed.

setInactiveText

```
virtual IInfoArea &setInactiveText(
                      const IString &inactiveText);

virtual IInfoArea &setInactiveText(
                      unsigned long inactiveText);
```

Sets the text that is used when the menu is inactive.

setLineCount

Condition: (2) OS/2 Only

```
virtual IInfoArea &setLineCount(unsigned long lines = 1);
```

Sets the number of text lines that the information area uses to display text. The default is 1 line.

setMissingText

```
virtual IInfoArea &setMissingText(
                      unsigned long missingText);

virtual IInfoArea &setMissingText(
                      const IString &missingText);
```

Sets the text that displays when the User Interface Class Library cannot find the information string.

setResourceLibrary

Condition: (1) Ignored on AIX

```
virtual IInfoArea &setResourceLibrary(
                    const char *resDLLName);
```

Condition: (1) Ignored on AIX

```
virtual IInfoArea &setResourceLibrary(
                    const IModuleHandle &resMod);
```

Sets the specified resource library. You can specify the library using:

- A module handle
- A .dll file
- A library name

setStringTableOffset

```
virtual IInfoArea &setStringTableOffset(long newOffset = 0);
```

Sets the offset to be used to calculate the information string ID.

stopHandlingEventsFor

```
virtual IInfoArea &stopHandlingEventsFor(
                    IFrameWindow *frame);
```

Detaches the handler of the information area from the specified frame window.

stringTableOffset

```
virtual long stringTableOffset() const;
```

Returns the offset currently in use.

Figure 144. Inherited Public Members

Member function	Class
activeColor	IWindow
alignment	IStaticText
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IStaticText
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow

Figure 144. Inherited Public Members

Member function	Class
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableFillBackground	IStaticText
disableGroup	IControl
disableHalftone	IStaticText
disableNotification	INotifier
disableNotification	IWindow
disableStrikeout	IStaticText
disableTabStop	IControl
disableUnderscore	IStaticText
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow

Figure 144. Inherited Public Members

Member function	Class
enableFillBackground	IStaticText
enableGroup	IControl
enableHalftone	IStaticText
enableNotification	INotifier
enableNotification	IWindow
enableStrikeout	IStaticText
enableTabStop	IControl
enableUnderscore	IStaticText
enableUpdate	IWindow
fillColor	IStaticText
font	IWindow
foregroundColor	IWindow
foregroundColor	IStaticText
handle	IWindow
handleException	IWindow
hasFillBackground	IStaticText
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHalftone	IStaticText
isLayoutDistorted	IWindow
isShowing	IWindow
isStrikeout	IStaticText
isTabStop	IWindow
isTabStop	IControl
isUnderscore	IStaticText
isValid	IWindow

Figure 144. Inherited Public Members

Member function	Class
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
limit	IStaticText
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveSizeTo	IStaticText
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetFillColor	IStaticText
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IStaticText

Figure 144. Inherited Public Members

Member function	Class
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFillColor	IStaticText
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setLimit	IStaticText
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IStaticText
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

Condition: (2) OS/2 Only

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

Returns the minimum size of the information area, based on the number of text lines that the information area displays and the text limit. The default text limit is 80.

dispatchHandlerEvent

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

Calls the appropriate virtual function or functions to process an information area event.

informationFor

```
virtual IString informationFor(unsigned long menuId) const;
```

Returns the information string for the specified menu ID.

menuEnded

```
virtual Boolean menuEnded(IMenuEvent &menuEvent);
```

Places the inactive information string into the information area.

menuSelected

```
virtual Boolean menuSelected(IMenuEvent &menuEvent);
```

Places the corresponding information string into the static text frame extension.

Figure 145. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcLimitSize	IStaticText
calcMinimumSize	IWindow
calcMinimumSize	IStaticText
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier

Figure 145. Inherited Protected Members

Member function	Class
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	IStaticText
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow

Figure 145. Inherited Protected Members

Member function	Class
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 102. IKeyboardEvent

Derivation

IBase
  IVBase
    IEvent
      IKeyboardEvent

Header File: ikeyevt.hpp

Objects of the IKeyboardEvent class represent 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, isScanCode, and isVirtual respectively, to do so. The library provides similar functions to check if the **Shift** keys, **Ctrl** keys, or **Alt** key is pressed.

If, however, you are within the callback function IKeyboardHandler::characterKeyPress (see page 457), you do not need to check the character code value with isCharacter. The same is true for the scan code value from IKeyboardHandler::scanCodeKeyPress (see page 457) and the virtual key value from IKeyboardHandler::virtualKeyPress (see page 457).

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 its owner window.

Constructors

```
public:
IKeyboardEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IKeyboardHandler::dispatchHandlerEvent (see page 457) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

© Copyright IBM Corp. 1994

character
```
char character() const;
```

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 (see page 450). If you expect double-byte data, use IKeyboardEvent::mixedCharacter (see page 451).

isAltDown
```
Boolean isAltDown() const;
```

Returns whether the **Alt** key is being held down.

isCharacter
```
Boolean isCharacter() const;
```

Returns whether the event represents use of a data key.

isComposite
```
Boolean isComposite() const;
```

Returns whether the character is formed by combining this key with the previous nonescaping key.

isCtrlDown
```
Boolean isCtrlDown() const;
```

Returns whether the **Ctrl** key is being held down.

isForComposite
```
Boolean isForComposite() const;
```

Returns whether the character is a nonescaping key, such as an accent, which the user intends to combine with the next character.

isInvalidComposite
```
Boolean isInvalidComposite() const;
```

Returns whether the character is not valid to be combined with the previous nonescaping key.

isRepeat
```
Boolean isRepeat() const;
```

Returns whether a key has been previously pressed and is being held down.

isScanCode
```
Boolean isScanCode() const;
```

Returns whether a hardware scan code is available.

isShiftDown
```
Boolean isShiftDown() const;
```

Returns whether the **Shift** key is being held down.

isUncombined
```
Boolean isUncombined() const;
```

Returns whether this key is being pressed without any other keys being pressed.

isUpTransition
```
Boolean isUpTransition() const;
```

Returns whether a previously pressed key is being released.

isVirtual
```
Boolean isVirtual() const;
```

Returns whether the event represents use of a virtual key.

mixedCharacter
```
IString mixedCharacter() const;
```

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 (see page 450).

Note: For portability, when working with DBCS characters, consider using IEditVerifyHandler (see page 253) instead of IKeyboardHandler (see page 455).

repeatCount
```
unsigned long repeatCount() const;
```

Returns the number of repeated keys combined into this event.

scanCode
```
unsigned long scanCode() const;
```

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 (see page 451).

virtualKey

```
VirtualKey virtualKey() const;
```

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 (see page 451). The enumeration VirtualKey (see page 130) provides the return values.

Figure 146. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Enumerations

VirtualKey

```
public:
enum 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,
endDrag,
firstDBCS,
lastDBCS,
firstUser,
lastUser,
other,
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 (see page 452) can return. The library provides enumerators for a generic keyboard, not just an IBM keyboard.

Presentation Manager Information: 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.

Chapter 103. IKeyboardHandler

Derivation

IBase
  IVBase
    IHandler
      IKeyboardHandler

Header File: ikeyhdr.hpp

Objects of the IKeyboardHandler class process all types of keyboard events. Some types of key processing, such as peeking at or modifying key input prior to its being stored in a control, are better handled by IEditVerifyHandler (see page 253).

You can attach IKeyboardHandlers to the following controls:

- ICheckBox (see page 71)
- IComboBox (see page 108)
- IContainerControl (see page 149)
- IEntryField (see page 256)
- IListBox (see page 458)
- IMultiLineEdit (see page 557)
- INotebook (see page 579)
- IProgressIndicator (see page 679)
- IRadioButton (see page 707)
- ISlider (see page 792)
- ISpinButton (see page 803)
- IBaseSpinButton (see page 32)
- INumericSpinButton (see page 612)
- ITextSpinButton (see page 878)

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 (see page 403) to pass this window to the keyboard handler.

When the keyboard handler receives a keyboard event, it creates an object of IKeyboardEvent (see page 449) and routes that object to the appropriate IKeyboardHandler 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 understand since 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:

© Copyright IBM Corp. 1994

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 (see page 946) to process the keyboard event. This call might cause the keyboard event to be dispatched to the focus window's owner window.

Portability Considerations: Some types of key processing, such as peeking at or modifying key input prior to its being stored in a control, are best handled in a portable way by using IEditVerifyHandler. See that class for details.

Constructors

```
public:
IKeyboardHandler();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

Public Members

Figure 147. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

characterKeyPress
```
virtual Boolean characterKeyPress(IKeyboardEvent &event);
```

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. This function can only be called for key-down transitions or if the user holds down a key.

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

If a keyboard event is received, the appropriate virtual function is called.

key
```
virtual Boolean key(IKeyboardEvent &event);
```

Implemented by derived classes to handle any and all unprocessed key events. This functions lets you process every keystroke as it is entered as well as key-down and key-up transitions.

scanCodeKeyPress
```
virtual Boolean scanCodeKeyPress(IKeyboardEvent &event);
```

Implemented by derived classes to handle scan code key presses. When using this callback function, you do not have to verify the character code value using IKeyboardEvent::isScanCode. This function can only be called for key-down transitions or if the user holds down a key.

virtualKeyPress
```
virtual Boolean virtualKeyPress(IKeyboardEvent &event);
```

Implemented by derived classes to handle virtual key presses. When using this callback function, you do not have to verify the character code value using IKeyboardEvent::isVirtual. This function can only be called for key-down transitions or if the user holds down a key.

Figure 148. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 104. IListBox

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IListBox

Header File: ilistbox.hpp

Objects of the IListBox class create and manage list box control windows. 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, if required.

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:

- IFocusHandler (see page 307)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)
- ISelectHandler (see page 764)

Portability Considerations: 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 Presentation Manager list box control, on the other hand, accesses each list box entry via a unique handle.

Motif Information: The User Interface Class Library implements the IListBox class using the XmList widget. Motif application developers should note that the IListBox 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

© Copyright IBM Corp. 1994

consisting of an XmScrolledWindow parent with an XmList child. The IListBox 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 IListBox 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.

Presentation Manager Information: When you add an IListBox as a child window of a canvas and use ISetCanvas, ISplitCanvas, or IMultiCellCanvas, you must give it the style IListBox::noAdjustPosition (see page 477).

Nested Classes

IListBox::Cursor
IListBox::Style

Constructors

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

public:
IListBox(unsigned long id, IWindow *parent);

public:
IListBox(const IWindowHandle &handle);
```

You can create a list box control object using the following:

- The parent window, owner window, optional size and location, and optional style arguments
- The parent window and a list box control ID
- The handle of an existing list box control

id The ID of a list box control.

parent The parent window.

owner The owner window.

initial (Optional) 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.

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

handle A window handle of an existing list box control.

Public Members

add

```
virtual IListBox &add(const char *item, Cursor &cursor);

virtual unsigned long add(
                      unsigned long index,
                      const char *text);

virtual unsigned long add(
                      unsigned long index,
                      const char * const *itemList,
                      unsigned long count = 1);

virtual unsigned long add(
                      unsigned long index,
                      const IResourceId &item);

virtual IListBox &add(
                    const IResourceId &text,
                    Cursor &cursor);
```

Inserts one of the following, depending on which form of the virtual member function you use:

- One or more lines of text at the specified index in the list box and returns the index of the first item inserted
- An item at the cursor position and sets the cursor to the item inserted

text The resource ID of the text string or the text string to insert in the list box.

cursor The list box cursor (see page 474).

index The index after which a text string or array of strings is inserted into the list box.

itemList An array of strings to be inserted into the list box.

count The number of strings in the itemList.

addAscending

```
virtual unsigned long addAscending(const char *text);

virtual unsigned long addAscending(const IResourceId &item);
```

Inserts the line of text in ascending sort order and returns the index of the item inserted.

text The text string to insert into the list box.

item The resource ID of the string to insert in the list box.

addAsFirst

```
virtual IListBox &addAsFirst(
                    const IResourceId &text,
                    Cursor &cursor);

virtual unsigned long addAsFirst(const char *text);

virtual unsigned long addAsFirst(const IResourceId &item);

virtual IListBox &addAsFirst(
                    const char *item,
                    Cursor &cursor);
```

Inserts one of the following, depending on which form of the virtual function you use:

- A line of text at the front of the list box and returns the index of the item inserted
- An item as the first item and puts the cursor on that item

text The resource ID of the text string or actual text string to insert into the list box.

cursor A list box cursor object) (see page 474).

item The resource ID of the text string or the text string.

addAsLast

```
virtual IListBox &addAsLast(
                    const IResourceId &text,
                    Cursor &cursor);

virtual unsigned long addAsLast(const char *text);

virtual unsigned long addAsLast(const IResourceId &item);

virtual IListBox &addAsLast(
                    const char *item,
                    Cursor &cursor);
```

Inserts one of the following, depending on which form of the virtual function you use:

- A line of text at the end of the list box and returns the index of the item inserted
- An item as the last item and puts the cursor on that item

text The resource ID or the text string to insert into the list box.

cursor A list box cursor (see page 474).

item The resource ID of the text string or the text string.

addAsNext

```
virtual IListBox &addAsNext(
                    const char *item,
                    Cursor &cursor);

virtual IListBox &addAsNext(
                    const IResourceId &text,
                    Cursor &cursor);
```

Adds the item as the next item in the list box and sets the cursor on the added item.

text The inserted text string's ID.

cursor A list box object.

item The text string.

addDescending

```
virtual unsigned long addDescending(
                      const IResourceId &item);

virtual unsigned long addDescending(const char *text);
```

Inserts the line of text in descending sort order and returns the index of the item inserted.

item The resource ID of the text string.

text The text string to be inserted into the list box.

backgroundColor

Condition: (2) OS/2 Only

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

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

count

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

Returns the count of the number of items in the list box.

Note: You can use IListBox::count and isSelected to process all of the selected list box items. However, it is easier to use the cursor facility provided by IListBox::Cursor (see page 474).

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 477) unless you have changed the style using setDefaultStyle (see page 467).

deselect

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

Removes the selection state from an item.

index The index of the item.

deselectAll

```
virtual IListBox &deselectAll();
```

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

disableDrawItem

Condition: (2) OS/2 Only

```
virtual IListBox &disableDrawItem();
```

Disables the style drawItem (see page 477) for the list box.

disableExtendedSelect

```
virtual IListBox &disableExtendedSelect();
```

Disables the style extendedSelect (see page 477) for the list box.

disableMultipleSelect

```
virtual IListBox &disableMultipleSelect();
```

Disables the style multipleSelect (see page 477) for the list box.

disableNoAdjustPosition

Condition: (1) Ignored on AIX

```
virtual IListBox &disableNoAdjustPosition();
```

Disables the style noAdjustPosition (see page 477) for the list box.

Motif Information: The AIX release does not process the noAdjustPosition style.

elementAt

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

Returns the item's string at the cursor.

enableDrawItem

Condition: (2) OS/2 Only

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

Enables or disables the style drawItem (see page 477) for the list box.

Motif Information: The AIX release does not support this function.

enableExtendedSelect

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

Enables or disables the style extendedSelect (see page 477) for the list box.

enableMultipleSelect

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

Enables or disables the style multipleSelect (see page 477) for the list box.

enableNoAdjustPosition

Condition: (1) Ignored on AIX

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

Enables or disables the style noAdjustPosition (see page 477) for the list box.

Motif Information: The AIX release does not process the noAdjustPosition style.

enableNotification

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

Enables or disables notification.

isDrawItem

Condition: (2) OS/2 Only

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

If the style drawItem (see page 477) is set, true is returned.

isEmpty

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

If the list box is empty, true is returned.

isExtendedSelect

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

If the style extendedSelect (see page 477) is set, true is returned.

isHorizontalScroll

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

If the style horizontalScroll (see page 477) is set, true is returned.

isMultipleSelect

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

If the style multipleSelect (see page 477) is set, true is returned.

isNoAdjustPosition

Condition: (1) Ignored on AIX

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

If the style noAdjustPosition (see page 477) is set, true is returned.

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

isSelected

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

Returns the selection state of an item.

Note: You can use IListBox::count and isSelected to process all of the selected list box items. However, it is easier to use the cursor facility provided by IListBox::Cursor (see page 474).

itemHandle

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

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

itemHeight

Condition: (3) AIX only

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

Returns the height, in pixels, of a list box item.

itemText

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

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

locateText

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

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 (see page 472). If no match is found, this function returns notFound. If there is an error, an exception is thrown.

index The search starts after the index. If you specify a search index of first (the default), the entire list is searched.

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

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

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

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

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

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.

remove
```
virtual unsigned long remove(unsigned long index);
```

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.

removeAll
```
virtual IListBox &removeAll();
```

Removes all items from the list box.

removeAt
```
virtual IListBox &removeAt(Cursor &cursor);
```

Removes the item at the cursor and places the cursor at the next available valid item.

replaceAt
```
virtual IListBox &replaceAt(
                    const IResourceId &text,
                    const Cursor &cursor);

virtual IListBox &replaceAt(
                    const char *item,
                    const Cursor &cursor);
```

Replaces the item's string at the cursor.

text The resource ID of the text string to replace in the list box.

cursor A list box cursor.

item The text string.

select

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

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 that is:

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.

selectAll

```
virtual IListBox &selectAll();
```

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

selection

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

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.

setDefaultStyle

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

Sets the default style for all subsequent list boxes.

style Use the styles provided by IListBox::Styles (see page 477) to specify the default list box style.

setHeight

```
virtual IListBox &setHeight(unsigned long newHeight);
```

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

Note: This function is obsolete. Use IListBox::setItemHeight instead.

setItemHandle

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

Sets the handle of the specified list box item.

setItemHeight

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

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

setItemText

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

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

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

setMinimumCharacters

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

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.

setMinimumRows

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

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

setTop

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

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

show

Condition: (3) AIX only

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

Makes the window visible.

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

top

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

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.

topHandle

Condition: (3) AIX only

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

Returns the top manager widget of the list box. The IWindowHandle (see page 960) returned is used for sizing, positioning, and managing the list box.

Figure 149. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl

Figure 149. Inherited Public Members

Member function	Class
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow

Figure 149. Inherited Public Members

Member function	Class
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow

Figure 149. Inherited Public Members

Member function	Class
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size of the list box.

The size dimensions are:

- For width, the average character width multiplied by the number of characters set by calling setMinimumCharacters. If you do not call this function, 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.

passEventToOwner

Condition: (3) AIX only

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

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

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

unregisterCallbacks

Condition: (3) AIX only

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

Called during destruction of the IListBox window. This function unregisters the Motif callbacks for the underlying widget.

Figure 150. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow

Figure 150. Inherited Protected Members

Member function	Class
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow

Figure 150. Inherited Protected Members

Member function	Class
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

SearchType

```
public:
enum 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.

Chapter 105. IListBox::Cursor

Derivation

IBase
  IVBase
    IListBox::Cursor

Header File: ilistbox.hpp

The nested class IListBox::Cursor defines objects that you can use to traverse the items in a list box. In the same way that you can use a cursor to traverse the objects in a collection, you can use a cursor to traverse 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.

Constructors

```
public:
Cursor(
   const IListBox &listbox,
   Filter type = selectedItems);
```

You can create objects of this class by providing:

listbox Reference to a list box control.

type A filter type. Use the enumeration Filter (see page 476) to specify this parameter. The default is selectedItems.

Public Members

asIndex

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

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

invalidate

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

Flags this cursor as not valid.

© Copyright IBM Corp. 1994

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

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

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

Points to the first list box item and validates the cursor.

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

Points to the specified item and validates the cursor.

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

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

Points to the last list box item and validates the cursor.

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

Points to the next item in the list box. If no more items exist, this function invalidates the cursor.

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

Points to the previous item in the list box. If no previous items exist, this function invalidates the cursor.

Figure 151. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

Filter

```
public:
enum Filter
  {
  selectedItems,
  allItems,
  };
```

Use these enumerators to specify which items you want filtered:

selectedItems
Filters only items that are selected.

allItems
Filters all items.

Chapter 106. IListBox::Style

Derivation

IBase
  IBitFlag
    IListBox::Style

Header File: ilistbox.hpp

The nested class IListBox::Style provides a set of valid list box styles for the static member functions of IListBox (see page 458). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

classDefaultStyle
Specifies the original default style for this class, which is horizontalScroll | noAdjustPosition | IWindow::visible.

Note: If you do not specify either the extendedSelect or multipleSelect style, the list box defaults to a single-selection list box.

drawItem
Specifies that list box items can draw graphics. Typically, you use this style to display bit maps.

extendedSelect
Specifies a selection optimized for the selection of a single object. A user can extend selection to more than one object, if required.

horizontalScroll
Specifies horizontal scrolling of the list box.

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

noAdjustPosition
Specifies that the list box control be drawn to the specified size. This can cause an item to be only partially shown.

Note: When you add an IListBox as a child window of a canvas, you must give it this style.

© Copyright IBM Corp. 1994

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

Public Members

Figure 152. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 153. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 107. IMenu

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IMenu

Header File: imenu.hpp

The IMenu class is an abstract base class for all menu classes, including:

- IMenuBar (see page 499)
- IPopUpMenu (see page 659)
- ISubMenu (see page 850)
- ISystemMenu (see page 867)

It provides functions to add, replace, delete, check, and disable menu items in the menu. This class also provides accessor functions to retrieve:

- IMenuItem objects in the menu
- The number of menu items in the menu
- The bounding rectangle for menu items in the menu

This class also provides color functions to query and set colors in the menu.

Motif Information: 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

Presentation Manager Information: 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.

Nested Classes

IMenu::Cursor
IMenu::Style

© Copyright IBM Corp. 1994

Constructors

```
protected:
IMenu();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

Public Members

add

```
virtual IMenu &add(
                  IMenuItem &menuItem,
                  IMenu::Cursor &cursor);
```

Adds the specified menu item at the specified cursor position by pushing down everything after the cursor.

addAsNext

```
virtual IMenu &addAsNext(
                  IMenuItem &menuItem,
                  IMenu::Cursor &cursor);
```

Adds the specified menu item as the next item and sets the specified cursor on it.

addAt

```
virtual IMenu &addAt(
                  IMenuItem &menuItem,
                  IMenu::Cursor &cursor);
```

Adds the specified menu item at the specified cursor position by pushing down everything after the cursor.

Note: This function is obsolete. Use IMenu::add instead.

Motif Information: The Motif resource XmNpositionIndex is set to position a menu item.

addBitmap

```
virtual IMenu &addBitmap(
                  unsigned long newItemId,
                  const IBitmapHandle &itemBitmap,
                  unsigned long intoSubmenuId = 0);

virtual IMenu &addBitmap(
                  unsigned long newItemId,
                  const IResourceId &bitmapResId,
                  unsigned long intoSubmenuId = 0);

virtual IMenu &addBitmap(
                  unsigned long newItemId,
```

```
                unsigned long bitmapResId,
                unsigned long intoSubmenuId = 0);
```

Adds a bit-map menu item as the last item in a menu or submenu.

newItemId
The new menu item to add to a menu or submenu.

bitmapResId
The bit map to load for the button face of the new menu item. This parameter accepts the resource ID of a bit map to load.

itemBitmap
The bit map to load for the button face of the new menu item. This parameter accepts a currently available bit-map handle.

intoSubmenuId
The submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif Information: 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:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

addItem

```
virtual IMenu &addItem(
                IMenuItem &menuItem,
                unsigned long intoSubmenuId = 0);
```

Adds a menu item, represented by an IMenuItem (see page 512), as the last item in a menu or submenu.

menuItem
A new menu item to add to a menu or submenu.

intoSubmenuId
A submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif Information: 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:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

addNextAt

```
virtual IMenu &addNextAt(
                  IMenuItem &menuItem,
                  IMenu::Cursor &cursor);
```

Adds the specified menu item as the next item and sets the specified cursor on it.

Note: This function is obsolete. Use IMenu::addAsNext instead.

Motif Information: In X/Motif, the resource XmNpositionIndex is set to position a menu item other than the last item.

addSeparator

```
virtual IMenu &addSeparator(
                  unsigned long intoSubmenuId = 0);

virtual IMenu &addSeparator(
                  unsigned long newItemId,
                  unsigned long intoSubmenuId);
```

Adds a separator 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.

newItemId
: New menu item to add to a menu or submenu.

intosubmenuId
: The submenu to add the new menu item to.

addSubmenu

```
virtual IMenu &addSubmenu(unsigned long itemId);

virtual IMenu &addSubmenu(
                  unsigned long itemId,
                  const IResourceId &submenuResId);
```

Creates a submenu and adds it to the menu item.

itemId
: The identifier of the menu item to add the submenu to. The menu item must already be in the menu.

submenuResId
: The resource ID 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.

addText

```
virtual IMenu &addText(
                  unsigned long newItemId,
                  const char *itemText,
                  unsigned long intoSubmenuId = 0);

virtual IMenu &addText(
                  unsigned long newItemId,
                  const IResourceId &textResId,
                  unsigned long intoSubmenuId = 0);
```

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.

newItemId
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.

itemText
The text to load for the button face of the new menu item. This parameter accepts a currently available text string.

intoSubmenuID
The submenu to add the new menu item to. If you specify 0, the new item is added to the menu bar.

Motif Information: 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:

- CascadeButton widgets or gadgets
- ToggleButton widgets or gadgets
- Label widgets or gadgets
- Separator widgets or gadgets

backgroundColor

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

Returns the background color value of the menu area, or the default if no color for the area has been set.

checkItem

```
virtual IMenu &checkItem(
                  unsigned long itemId,
                  Boolean check = true);
```

Places a select state indicator to the left of the specified item.

itemId
The menu item to check or uncheck.

check
An optional Boolean parameter that specifies whether the menu item should be checked.

Note: You cannot check the menu bar.

Motif Information: The select state indicator is a square object to the left of the specified item.

In X/Motif, the resources XmNset and XmNindicatorOn are set to check or uncheck a menu item.

Presentation Manager Information: The select state indicator is a check mark to the left of the specified item.

cursor

```
Cursor cursor(
         unsigned long itemId,
         unsigned long inSubmenuId = 0) const;
```

Returns an object of the nested class IMenu::Cursor (see page 495) 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*.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 497) unless you have changed it using setDefaultStyle (see page 490).

deleteAt

```
virtual IMenu &deleteAt(IMenu::Cursor &cursor);
```

Deletes the menu item at the position of the specified cursor and sets the cursor to the previous menu item.

deleteItem

```
virtual IMenu &deleteItem(unsigned long itemId);
```

Deletes the specified item from the menu.

disabledBackgroundColor

```
virtual IColor disabledBackgroundColor() const;
```

Returns the disabled background color value of the menu area, or the default if no color for the area has been set.

disabledForegroundColor

```
virtual IColor disabledForegroundColor() const;
```

Returns the disabled foreground color value of the menu area, or the default if no color for the area has been set.

disableItem

```
virtual IMenu &disableItem(unsigned long itemId);
```

Makes a menu item unselectable.

Motif Information: In X/Motif, the resource XmNsensitive is set by invoking XtSetSensitive to enable or disable a menu item.

elementAt

```
virtual IMenuItem elementAt(
                    const IMenu::Cursor &cursor) const;
```

Returns an IMenuItem that describes the menu item at the cursor position.

enableItem

```
virtual IMenu &enableItem(
                  unsigned long itemId,
                  Boolean enable = true);
```

Makes a menu item selectable.

itemId
The identifier of the menu item to be enabled or disabled.

check
(Optional) Specifies whether the menu item should be enabled.

Motif Information: In X/Motif, the resource XmNsensitive is set by invoking XtSetSensitive to enable or disable a menu item.

enableNotification

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

Causes the menu to send notifications to any observer objects.

foregroundColor

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

Returns the foreground color value of the menu area, or the default if no color for the area has been set.

hiliteBackgroundColor

```
virtual IColor hiliteBackgroundColor() const;
```

Returns the highlight background color value of the menu area, or the default if no color for the area has been set.

hiliteForegroundColor

```
virtual IColor hiliteForegroundColor() const;
```

Returns the highlight foreground color value of the menu area, or the default if no color for the area has been set.

isItemChecked

```
Boolean isItemChecked(unsigned long itemId) const;
```

If the specified menu item is checked, true is returned.

Motif Information: In X/Motif, the resources XmNset and XmNindicatorOn are queried to determine if a menu item is checked.

isItemDisabled

```
Boolean isItemDisabled(unsigned long itemId) const;
```

If the specified menu item is nonselectable, true is returned.

Note: This function is obsolete. Use IMenu::isItemEnabled instead.

Motif Information: In X/Motif, the library queries the resource XmNsensitive to determine if a menu item is nonselectable.

isItemEnabled

```
Boolean isItemEnabled(unsigned long itemId) const;
```

If the specified menu item is selectable, true is returned.

isVerticalFlip

```
Boolean isVerticalFlip() const;
```

Queries whether the style verticalFlip (see page 497) is set.

Note: This function is obsolete. There is no replacement function.

Motif Information: AIX does not support this member function.

itemRect

```
IRectangle itemRect(unsigned long itemId) const;
```

Returns the bounding rectangle of a menu item.

Presentation Manager Information: The bounding rectangle is the rectangle returned by sending the message MM_QUERYITEMRECT.

menuItem

```
IMenuItem menuItem(unsigned long itemId) const;
```

Returns an IMenuItem (see page 512) object. If the menu does not have a menu item with an ID equal to *itemId*, an exception is thrown.

menuItem will throw an exception if the menu does not contain an item with an ID equal to what you pass in. A try-catch block will let you know when an item does not exist.

numberOfItems

```
unsigned long numberOfItems(
                unsigned long forSubmenuId = 0) const;
```

Returns the number of menu items for the specified submenu. If you specify 0, the count is returned for the IMenu object.

Motif Information: In X/Motif, the library queries the resource XmNnumChildren to determine the number of items in a menu or submenu.

removeConditionalCascade

Condition: (1) Ignored on AIX

```
virtual IMenu &removeConditionalCascade(
                  unsigned long itemWithSubmenuId);
```

Removes conditional cascade behavior from the specified menu item.

itemWithSubmenuId
: The menu item from which to remove conditionally cascade. If you specify an *itemWithSubmenuId* that does not exist or does not have an associated submenu, an exception is thrown.

Motif Information: X/Motif does not support conditionally cascading menus. This member function returns *this not false. The function return type is IMenu&, thus *this is returned.

removeSubmenu

```
virtual IMenu &removeSubmenu(
                unsigned long itemWithSubmenuId);
```

Removes the menu item's submenu indicator and destroys the submenu.

itemWithSubmenuId
The ID of the item whose submenu is to be removed.

removeSubmenuAt

```
virtual IMenu &removeSubmenuAt(IMenu::Cursor &cursor);
```

Removes the menu item and submenu indicator at the specified cursor position. Typically, you use this function to remove a cascading menu.

resetBackgroundColor

```
virtual IMenu &resetBackgroundColor();
```

Resets the background color by undoing a previously set color.

resetDisabledBackgroundColor

```
virtual IMenu &resetDisabledBackgroundColor();
```

Resets the disabled background color by undoing a previously set color.

resetDisabledForegroundColor

```
virtual IMenu &resetDisabledForegroundColor();
```

Resets the disabled foreground color by undoing a previously set color.

resetForegroundColor

```
virtual IMenu &resetForegroundColor();
```

Resets the foreground color by undoing a previously set color.

resetHiliteBackgroundColor

```
virtual IMenu &resetHiliteBackgroundColor();
```

Resets the highlight background color by undoing a previously set color.

resetHiliteForegroundColor

```
virtual IMenu &resetHiliteForegroundColor();
```

Resets the highlight foreground color by undoing a previously set color.

selectItem

```
virtual IMenu &selectItem(unsigned long itemId);
```

Selects a menu item.

Motif Information: In X/Motif, the resource XmNset is set to select a menu item.

setBackgroundColor

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

Sets the background color.

setBitmap

```
virtual IMenu &setBitmap(
                  unsigned long menuItemId,
                  const IResourceId &newBitmapResId);

virtual IMenu &setBitmap(
                  unsigned long menuItemId,
                  const IBitmapHandle &bitmapHandle);

virtual IMenu &setBitmap(
                  unsigned long menuItemId,
                  unsigned long newBitmapResId);
```

Causes a menu item to display a bit map. The function expects the bit map to be in the default resource library.

menuItemId
: The menu item to add the bit map to.

newBitmapResId
: The bit map to display. This parameter accepts the resource ID of a bit map.

bitmapHandle
: The bit map to display. This parameter accepts a currently available bit-map handle.

setConditionalCascade

Condition: (1) Ignored on AIX

```
virtual IMenu &setConditionalCascade(
                  unsigned long itemWithSubmenuId,
                  unsigned long defaultItemId);
```

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:

- The specified items for *itemWithSubmenuId* or *defaultItemId* do not exist.
- The specified item for *itemWithSubmenuId* does not have a submenu attached.

itemWithSubmenuId
The menu item to have its submenu set as a conditionally cascading menu.

defaultItemId
The default selection.

Motif Information: X/Motif does not support conditionally cascading menus. This member function returns *this not false. The function return type is IMenu&, thus *this is returned.

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

Sets the default style for all subsequent menus.

setDisabledBackgroundColor
```
virtual IMenu &setDisabledBackgroundColor(
                  const IColor &color);
```

Sets the disabled background color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

setDisabledForegroundColor
```
virtual IMenu &setDisabledForegroundColor(
                  const IColor &color);
```

Sets the disabled foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

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

Sets the foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

setHiliteBackgroundColor
```
virtual IMenu &setHiliteBackgroundColor(
                  const IColor &color);
```

Sets the hilite background color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

setHiliteForegroundColor
```
virtual IMenu &setHiliteForegroundColor(
                  const IColor &color);
```

Sets the hilite foreground color to the indicated color. The menu area is identified by a system-defined presentation parameter value.

setItem

```
virtual IMenu &setItem(const IMenuItem &menuItem);
```

Replaces a specified menu item's style and representation.

setSubmenu

```
virtual IMenu &setSubmenu(
                  unsigned long itemId,
                  const IResourceId &submenuResId);
```

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.

setText

```
virtual IMenu &setText(
                  unsigned long menuItemId,
                  const char *newText);

virtual IMenu &setText(
                  unsigned long menuItemId,
                  const IResourceId &newTextResId);
```

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.

setVerticalFlip

```
IMenu &setVerticalFlip(Boolean flip = true);
```

Displays the menu above the menu bar, if possible.

Note: This function is obsolete. There is no replacement function.

Motif Information: AIX does not support this member function.

uncheckItem

```
virtual IMenu &uncheckItem(unsigned long itemId);
```

Removes the select state indicator from a specified menu item.

Motif Information: In X/Motif, the resources XmNset and XmNindicatorOn are set to check or uncheck a menu item.

Figure 154. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow

Figure 154. Inherited Public Members

Member function	Class
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow

Figure 154. Inherited Public Members

Member function	Class
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow

Figure 154. Inherited Public Members

Member function	Class
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 155. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 108. IMenu::Cursor

Derivation

IBase
  IVBase
    IMenu::Cursor

Header File: imenu.hpp

Objects of the IMenu::Cursor class create and manage the cursor for menu items. You can add or update items anywhere in your menu structure using this class.

Constructors

```
public:
Cursor(const IMenu &menu, unsigned long forSubmenuId = 0);
```

You can only construct objects of this class from an object of IMenu (see page 479) and a submenu ID. The submenu defaults to 0 for the top-level menu items.

Public Members

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

Flags the cursor as invalid.

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

Queries whether this cursor points to a valid menu item.

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

Points to the first menu item. If successful, true is returned.

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

Points to the last menu item. If successful, true is returned.

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

© Copyright IBM Corp. 1994

Points to the next menu item. If there is none, the cursor is invalidated. If successful, true is returned.

setToPrevious

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

Points to the previous menu item. If there is none, the cursor is invalidated. If successful, true is returned.

Figure 156. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 109. IMenu::Style

Derivation

IBase
  IBitFlag
    IMenu::Style

Header File: imenu.hpp

The nested class IMenu::Style provides a set of valid menu styles for the static member functions of the class IMenu (see page 479). You can use these styles in conjunction with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

classDefaultStyle
Specifies the original default style for this class, which is noStyle.

verticalFlip
Displays the menu above the menu bar, if possible. Otherwise, the menu is displayed below the menu bar.

Note: This style is obsolete. There is no replacement style.

noStyle
Specifies that no style applies to the menu.

Motif Information: Motif does not support the verticalFlip style. There is no way to control where menus are mapped. However, the Motif MenuShell provides default verticalFlip behavior in cases where menus are potentially mapped outside the current screen boundary.

Public Members

Figure 157. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*

Figure 157. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 158. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 110. IMenuBar

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IMenu
          IMenuBar

Header File: imenubar.hpp

Objects of the IMenuBar class create a menu bar for frame windows.

Motif Information: The User Interface Class Library adds an X/Motif callback routine to all menu bars for processing selected menu bar items.

Nested Classes

IMenuBar::Style

Constructors

```
public:
IMenuBar(const IResourceId &menuResId, IFrameWindow *owner);

public:
IMenuBar(
   IFrameWindow *owner,
   const Style &style = defaultStyle ( ));
```

You can construct objects of this class by:

- Creating 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 file.
- Creating an empty menu bar or a wrapper for an IFrameWindow (see page 352) menu bar. Use this constructor to create:
  - A menu bar with no entries, using the style IMenuBar::empty (see page 504).
  - A menu bar wrapper, using the style IMenuBar::wrapper (see page 504).

Note: You can add a menu bar to a frame window without using the IMenuBar class by specifying the style IFrameWindow::menuBar (see page 376).

© Copyright IBM Corp. 1994

Public Members

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

Returns the default style. The default style is classDefaultStyle (see page 504) unless you have changed it using setDefaultStyle (see page 500).

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

Sets the default style for all subsequent menu bars.

setMenu
```
virtual IMenuBar &setMenu(const IResourceId &menuResId);
```

Sets or changes the menu bar to the specified menu resource ID.

Figure 159. Inherited Public Members

Member function	Class
activeColor	IWindow
add	IMenu
addAsNext	IMenu
addAt	IMenu
addBitmap	IMenu
addItem	IMenu
addNextAt	IMenu
addSeparator	IMenu
addSubmenu	IMenu
addText	IMenu
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IMenu
borderColor	IWindow
characterSize	IWindow
checkItem	IMenu
childAt	IWindow
cursor	IMenu
defaultPushButton	IWindow

Figure 159. Inherited Public Members

Member function	Class
deleteAt	IMenu
deleteItem	IMenu
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IMenu
disabledForegroundColor	IWindow
disabledForegroundColor	IMenu
disableItem	IMenu
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
elementAt	IMenu
enable	IWindow
enableItem	IMenu
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IMenu
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IMenu
handle	IWindow
handleException	IWindow
hasFocus	IWindow

Figure 159. Inherited Public Members

Member function	Class
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IMenu
hiliteForegroundColor	IWindow
hiliteForegroundColor	IMenu
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isItemChecked	IMenu
isItemDisabled	IMenu
isItemEnabled	IMenu
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVerticalFlip	IMenu
isVisible	IWindow
itemProvider	IWindow
itemRect	IMenu
layoutAdjustment	IWindow
matchForMnemonic	IWindow
menuItem	IMenu
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
numberOfItems	IMenu
operator <<	IBase*
operator <<	IVBase*
owner	IWindow

Figure 159. Inherited Public Members

Member function	Class
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeConditionalCascade	IMenu
removeSubmenu	IMenu
removeSubmenuAt	IMenu
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBackgroundColor	IMenu
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledBackgroundColor	IMenu
resetDisabledForegroundColor	IWindow
resetDisabledForegroundColor	IMenu
resetForegroundColor	IWindow
resetForegroundColor	IMenu
resetHiliteBackgroundColor	IWindow
resetHiliteBackgroundColor	IMenu
resetHiliteForegroundColor	IWindow
resetHiliteForegroundColor	IMenu
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
selectItem	IMenu
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBackgroundColor	IMenu
setBitmap	IMenu
setBorderColor	IWindow
setConditionalCascade	IMenu

Figure 159. Inherited Public Members

Member function	Class
setDisabledBackgroundColor	IWindow
setDisabledBackgroundColor	IMenu
setDisabledForegroundColor	IWindow
setDisabledForegroundColor	IMenu
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setForegroundColor	IMenu
setHiliteBackgroundColor	IWindow
setHiliteBackgroundColor	IMenu
setHiliteForegroundColor	IWindow
setHiliteForegroundColor	IMenu
setId	IWindow
setInactiveColor	IWindow
setItem	IMenu
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setSubmenu	IMenu
setText	IMenu
setVerticalFlip	IMenu
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
uncheckItem	IMenu
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

registerCallbacks

Condition: (3) AIX only

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

Adds X/Motif callbacks for an XmMenuBar row column widget.

unregisterCallbacks

Condition: (3) AIX only

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

Removes X/Motif callbacks from a XmMenuBar row column widget.

Figure 160. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 111. IMenuBar::Style

Derivation

IBase
  IBitFlag
    IMenuBar::Style

Header File: imenubar.hpp

The nested class IMenuBar::Style provides a set of valid menu bar styles for the static member functions of the class IMenuBar (see page 499). You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IMenu::Style (see page 497)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IMenuBar::empty.

empty
Creates the menu with no items. You can dynamically populate the menu using functions from IMenu (see page 479) and ISubmenu (see page 850).

wrapper
Specifies the menu bar is a wrapper for an IFrameWindow (see page 352) menu bar, which you previously created using the style IFrameWindow::menuBar (see page 376).

Public Members

Figure 161. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 162. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 112. IMenuEvent

Derivation

IBase
  IVBase
    IEvent
      IMenuEvent

Header File: imenuevt.hpp

Objects of the IMenuEvent class provide information about a menu event for the event-handling functions of IMenuHandler (see page 508).

Constructors

```
public:
IMenuEvent(IEvent &event);
```

Although you can construct objects of this class, typically IMenuHandler::dispatchHandlerEvent (see page 510) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

menuItem

```
IMenuItem menuItem() const;
```

Returns an object of IMenuItem (see page 512) for the menu item. This function only returns an actual IMenuItem when called because a user selected a menu item. If this function is called during any of the other menu actions, an empty IMenuItem for *itemId* 0 is returned.

menuItemId

```
unsigned long menuItemId() const;
```

Returns the ID of the menu item. The returned menu item ID can be operated upon by IMenuHandler::menuEnded (see page 510), IMenuHandler::menuSelected (see page 511), and IMenuHandler::menuShowing (see page 511).

mousePosition

```
IPoint mousePosition() const;
```


© Copyright IBM Corp. 1994

Returns the mouse position at the time the menu event occurred. In Presentation Manager, the User Interface Class Library maps the mouse position to the menu's owner window's coordinates. In Motif, it is the coordinates of the IWindow over which the mouse click occurred. In either case, you may use the value returned as the parameter on IPopUpMenu::show.

Figure 163. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 113. IMenuHandler

Derivation

```
IBase
  IVBase
    IHandler
      IMenuHandler
```

Header File: imenuhdr.hpp

Objects of the IMenuHandler class process all types of menu events except menu command events, which are handled by the class ICommandHandler (see page 131). You can override IMenuHandler::menuShowing (see page 511) tp make changes via the ISubmenu object that is passed as a parameter. You can also use this class to display a pop-up menu.

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 (see page 403) to pass the appropriate frame window or control to the menu handler.

When the menu handler receives a menu event, it creates an IMenuEvent (see page 506) 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.

false 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 (see page 946) to process the menu event.

Constructors

```
public:
IMenuHandler();
```

You can only construct objects of this class by using the default constructor, which does not accept any parameters.

© Copyright IBM Corp. 1994

Public Members

handleEventsFor

Condition: (3) AIX only

```
virtual IMenuHandler &handleEventsFor(IWindow *window);
```

Attaches the menu handler to the specified IWindow object.

stopHandlingEventsFor

Condition: (3) AIX only

```
virtual IMenuHandler &stopHandlingEventsFor(
                        IWindow *window);
```

Detaches the menu handler from the specified IWindow object.

Figure 164. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

addSourceEmphasis

```
virtual void addSourceEmphasis(const IMenuEvent &menuEvent);
```

Calls IWindow::showSourceEmphasis (see page 942) which ignores it by default.

Presentation Manager Information: Only IContainerControl (see page 149) implements this function.

dispatchHandlerEvent

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

If a menu event is received, the appropriate virtual function is called.

makePopUpMenu

```
virtual Boolean makePopUpMenu(IMenuEvent &menuEvent);
```

Called when the user causes a context menu event to occur while on top of a control associated with the IMenuHandler. A context menu event is associated with a context or pop-up menu. Within the handler routine, you can either:

- 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 (see page 659). Code your override function to return true if your program constructs an IPopUpMenu; otherwise, code it to return false.

The pop-up menu terminates when IMenuHandler::menuEnded (see page 510) is called. If, at this point, you want the library to delete your IPopUpMenu object, call IWindow::setAutoDeleteObject (see page 938). In order to do so, you must create the IPopUpMenu with the global operator new.

IPopUpMenu::makePopUpMenu must return a pointer to the menu being shown. The menu can be:

- Off the heap, if you use the global operator new
- Off the stack
- Static

If the menu is off the heap, you can:

1. Declare an IPopUpMenu* from within makePopUpMenu, allocating memory through global operator new.
2. Set the last parameter in the constructor to true so the handler knows to delete the object after the menu is hidden.

You need to call IPopUpMenu::show (see page 660) within makePopUpMenu because IPopUpMenu::show allows you to specify where the menu appears.

menuEnded

```
virtual Boolean menuEnded(IMenuEvent &menuEvent);
```

Called when a submenu, such as a pull-down menu, cascading menu, or pop-up menu, terminates.

If your menu handler is attached to a pop-up menu that has setAutoDeleteObject set to true, you should return true if you override menuEnded.

menuSelected

```
virtual Boolean menuSelected(IMenuEvent &menuEvent);
```

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, pulldown or cascade) 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, MouseButton1)
- Using the **Alt** or **PF10** keys

  Note: These keys only highlight the first item on your menu bar and cause this function to be called.

IInfoArea (see page 441) uses this function to look up the appropriate help panel to display based on the highlighted menu item.

menuShowing

```
virtual Boolean menuShowing(
                  IMenuEvent &menuEvent,
                  ISubmenu &submenuAboutToShow);
```

Called before a submenu, such as a pull-down, cascading, or pop-up menu, is shown.

You can provide an object of the class ISubmenu (see page 850) to alter the menu items temporarily. If this function returns true, any change to the ISubmenu is undone when you call IMenuHandler::menuEnded (see page 510) to terminate the menu.

menuEvent
Current menu event object.

submenuAboutToShow
A submenu object to temporarily change the menu items.

removeSourceEmphasis

```
virtual void removeSourceEmphasis(
               const IMenuEvent &menuEvent);
```

Calls IWindow::hideSourceEmphasis (see page 929).

Figure 165. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 114. IMenuItem

Derivation

IBase
  IMenuItem

Header File: imnitem.hpp

Objects of the IMenuItem class represent the menu item structure. This class, which is a data class, provides the following:

- Style objects
- Attribute objects
- Functions to set the menu item
- Functions to query the menu item

These functions do not affect the menu unless you call IMenu::add or IMenu::set using the menu item object.

You can construct and add an IMenuItem to the menu by using the member functions of IMenu (see page 479) or ISubmenu (see page 850). You can also use IMenu::menuItem (see page 487) to do the following:

- Query an existing item in a menu
- Change the item's style and attribute
- Reset the menu item

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, such as when applying different actions to a menu item object when the user selects it from the menu bar as opposed to the pop-up menu, you must use different IDs for the menu items to distinguish whether they are on the menu bar or the pop-up menu.

Motif Information: All menu item IDs must be unique.

Presentation Manager Information: A menu item ID does not have to be unique. However, if you call a function such as IMenu::checkItem (see page 483) using the ID, the first menu item that has the ID is acted on.

© Copyright IBM Corp. 1994

Nested Classes

IMenuItem::Style
IMenuItem::Attribute

Constructors

```
public:
IMenuItem(
   unsigned long itemId,
   const Style &style = defaultStyle ( ),
   const Attribute &attribute = defaultAttribute ( ));
```

You can only construct objects of this class by creating a default menu item. This constructor creates an IMenuItem object with the specified styles and attributes.

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 bit-map menu item, first construct the IMenuItem, and then call setBitmap (see page 516) to specify the bit map to be used for display purposes.

After construction, you can add IMenuItems that have the styles separator (see page 523) or drawItem (see page 523) to a menu without having to call any of the set functions.

Public Members

attribute

```
unsigned long attribute() const;
```

Returns an unsigned long representing the menu item's attributes.

bitmap

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

Returns the bit-map handle of the item.

commandType

```
CommandType commandType() const;
```

Returns the event the menu item generates when selected. The event can be a command, systemCommand, or helpCommand.

defaultAttribute

```
static Attribute defaultAttribute();
```

Returns the default attribute. The default attribute is classDefaultAttribute unless you have changed it using setDefaultAttribute (see page 516).

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

Returns the default style. The default style is classDefaultStyle (see page 523) unless you have changed it using setDefaultStyle (see page 516).

id
```
unsigned long id() const;
```

Returns the ID associated with the menu item.

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

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.

isChecked
```
Boolean isChecked() const;
```

If the attribute IMenuItem::checked is set, true is returned.

isDisabled
```
Boolean isDisabled() const;
```

If the attribute IMenuItem::disabled is set, true is returned.

isDrawItem
Condition: (1) Ignored on AIX

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

If the style IMenuItem::drawItem (see page 523) is set, true is returned.

Motif Information: X/Motif does not support the style drawItem. This member function always returns false.

isFramed
Condition: (1) Ignored on AIX

```
Boolean isFramed() const;
```

If the attribute IMenuItem::framed is set, true is returned.

Motif Information: X/Motif does not support the framed attribute. This member function always returns false.

isHighlighted

Condition: (1) Ignored on AIX

```
Boolean isHighlighted() const;
```

If the attribute IMenuItem::highlighted is set, true is returned.

Motif Information: X/Motif does not support the highlighted attribute. This member function always returns false.

isNoDismiss

Condition: (1) Ignored on AIX

```
Boolean isNoDismiss() const;
```

If the attribute IMenuItem::noDismiss is set, true is returned.

Motif Information: X/Motif does not support the attribute noDismiss. This member function always returns false.

isSelectable

```
Boolean isSelectable() const;
```

If the menu item is selectable, true is returned.

isSeparator

```
Boolean isSeparator() const;
```

If the style IMenuItem::separator (see page 523) is set, true is returned.

layoutType

```
LayoutType layoutType() const;
```

Returns the menu item placement and layout.

setAttribute

```
IMenuItem &setAttribute(unsigned long newAttribute);
```

Sets the menu item's attributes from the specified unsigned long.

setBitmap

```
IMenuItem &setBitmap(unsigned long bitmapResId);

IMenuItem &setBitmap(const IBitmapHandle &menuItem);

IMenuItem &setBitmap(const IResourceId &bitmapResId);
```

Sets the bit map to be displayed. The implementation using an unsigned long parameter is the same as IResourceId (see page 736). The User Interface Class Library provides this implementation to resolve ambiguity.

The parameters are the following:

bitmapResId
: Resource ID of the bitmap to display as a menu item.

menuItem Handle of the bitmap to display as a menu item.

setChecked

```
IMenuItem &setChecked(Boolean checked = true);
```

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.

Motif Information: The state indicator is a square-shaped object displayed to the left of the menu item.

Presentation Manager Information: The state indicator is a check mark displayed to the left of the menu item.

setCommand

```
IMenuItem &setCommand(CommandType value);
```

Specifies what event to generate when the user selects the menu item. The event can be a command, systemCommand, or helpCommand.

setDefaultAttribute

```
static void setDefaultAttribute(const Attribute &attribute);
```

Sets the default attribute for all subsequent menu items.

setDefaultStyle

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

Sets the default style for all subsequent menu items.

setDisabled

```
IMenuItem &setDisabled(Boolean disabled = true);
```

Displays the menu item with unavailable-state emphasis, which means the user cannot select it.

setDrawItem

Condition: (1) Ignored on AIX

```
IMenuItem &setDrawItem(Boolean drawItem = true);
```

Causes the system to generates a draw item event when the item needs drawing. This enables the application to draw the menu item.

Motif Information: X/Motif does not support the drawItem style.

setFramed

Condition: (1) Ignored on AIX

```
IMenuItem &setFramed(Boolean framed = true);
```

Draws a box to enclose the menu item.

Motif Information: X/Motif does not support the framed attribute.

setHighlighted

Condition: (1) Ignored on AIX

```
IMenuItem &setHighlighted(Boolean highlighted = true);
```

Displays the menu item with selected-state emphasis.

Motif Information: X/Motif does not support the highlighted attribute.

setIndex

```
IMenuItem &setIndex(unsigned long index = IMenuItem::atEnd);
```

Sets the menu item ordinal position in its submenu. With this member function, you can add a menu item anywhere in the order.

setLayout

```
IMenuItem &setLayout(LayoutType value);
```

Defines the menu item placement and layout. Use values from the enumeration IMenuItem::LayoutType (see page 519) to specify the layout type and placement.

setNoDismiss

Condition: (1) Ignored on AIX

```
IMenuItem &setNoDismiss(Boolean noDismiss = true);
```

Forces the menu item to remain visible when selected.

Motif Information: X/Motif does not support the noDismiss attribute.

setSelectable
```
IMenuItem &setSelectable(Boolean selectable = true);
```

Enables the menu item for selection.

setSeparator
```
IMenuItem &setSeparator();
```

Makes the menu item a separator.

setStyle
```
IMenuItem &setStyle(unsigned long newStyle);
```

Sets the menu item's style from the specified unsigned long.

setSubmenuHandle
```
IMenuItem &setSubmenuHandle(
              const IWindowHandle &menuHandle);
```

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.

setText
```
IMenuItem &setText(const IResourceId &textResId);

IMenuItem &setText(const char *newText);
```

Sets the text to be displayed. You can use either the resource ID of the text or the actual text.

style
```
unsigned long style() const;
```

Returns an unsigned long representing the menu item's style.

submenuHandle
```
IWindowHandle submenuHandle() const;
```

Returns the submenu window handle for the item. If the item does not have a submenu, 0 is returned.

text
```
IString text() const;
```

Returns an IString representing the text to be displayed.

Figure 166. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Enumerations

CommandType

```
public:
enum CommandType
  {
  command,
  systemCommand,
  helpCommand,
  };
```

These enumerators specify the type of event generated when a user selects the menu item:

command
Generates an ICommandHandler::command (see page 133) event.

systemCommand
Generates an ICommandHandler::systemCommand (see page 134) event.

helpCommand
Generates a help message.

LayoutType

```
public:
enum LayoutType
  {
  normalLayout,
  splitLayout,
  splitWithSeparatorLayout,
  buttonSeparatorLayout,
  };
```

These enumerators specify the placement and layout of the menu item:

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 Information: X/Motif does not support the enumerators splitWithSeparatorLayout and buttonSeparatorLayout.

Chapter 115. IMenuItem::Attribute

Derivation

IBase
  IBitFlag
    IMenuItem::Attribute

Header File: imnitem.hpp

The nested class IMenuItem::Attribute provides a set of valid menu item attributes for the static member functions of the class IMenuItem (see page 512). All attributes are set independently of each other.

Attributes

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.

classDefaultAttribute
Specifies this attribute is a static variable set to noAttribute.

disabled
Displays the menu item with unavailable-state emphasis, which means the user cannot select it.

framed
Encloses the menu item in a box.

highlighted
Displays the menu item with selected-state emphasis.

noAttribute
Specifies that no attribute applies to the menu item.

noDismiss
If a user chooses a menu item in a submenu, keeps the submenu displayed.

Motif Information: The state indicator is a square-shaped object displayed to the left of the menu item.

X/Motif does not support the following attributes:

- framed
- highlighted
- noDismiss

© Copyright IBM Corp. 1994

Presentation Manager Information: The state indicator is a check mark displayed to the left of the menu item.

Public Members

Figure 167. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 168. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 116. IMenuItem::Style

Derivation

IBase
  IBitFlag
    IMenuItem::Style

Header File: imnitem.hpp

The nested class IMenuItem::Style provides a set of valid menu item styles to the static member functions of class IMenuItem (see page 512).

Styles

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 split or splitWithSeparator styles.

classDefaultStyle
Specifies the original default style for this class, which is noStyle.

drawItem
Enables applications to draw the menu item.

noStyle
Specifies that no style applies to the menu item.

postHelp
In OS/2 Presentation Manager, specifies that the menu item generate a help message that is issued to the previous focus window. Therefore, the postHelp style works just like a **Help** push button.

Note: You cannot use this style with the postSystemCommand style.

In Motif, specifies that the menu item is a help menu. When you specify this style on a menu item which appears in the menu bar, the menu item appears to the far right of the menu bar.

postSystemCommand
Specifies the menu item generate a system command message that can be handled by ICommandHandler::systemCommand (see page 134).

Note: You cannot use this style with the postHelp style.

separator
Specifies that the menu item is a separator.

© Copyright IBM Corp. 1994

split
Starts the menu item in a new row or column.

Note: You cannot use this style with the splitWithSeparator or buttonSeparator styles.

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 split or buttonSeparator styles.

unavailable
Sets unavailable-state emphasis for a menu item.

Motif Information: X/Motif does not support the following styles:

- drawItem
- buttonSeparator
- splitWithSeparator

Public Members

Figure 169. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 170. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 117. IMessageBox

Derivation

IBase
  IVBase
    IMessageBox

Header File: imsgbox.hpp

Objects of the IMessageBox class display the message boxes that are compliant with both the Common User Access (CUA) and the Motif style guides.

Nested Classes

IMessageBox::Style

Constructors

```
public:
IMessageBox(IWindow *aOwner);
```

You can only construct objects of this class from an object of the class IWindow (see page 924).

aOwner The IWindow object that becomes the owner of the new message box. The message box title defaults to the owner's title.

Public Members

setTitle

```
virtual IMessageBox &setTitle(const IResourceId &title);

virtual IMessageBox &setTitle(const char *title);
```

Sets the message box's title.

title You can specify the title as either of the following:

- A resource ID to load from a resource file
- A character string

show

```
virtual IMessageBox::Response show(
                              const char *message,
                              const Style &style,
                              unsigned long helpId = 0);

virtual IMessageBox::Response show(
```

© Copyright IBM Corp. 1994

```
                                const char *message,
                                Severity severity,
                                unsigned long helpId = 0);

virtual IMessageBox::Response show(
                                const IResourceId &message,
                                Severity severity,
                                unsigned long helpId = 0);

virtual IMessageBox::Response show(
                                const IResourceId &message,
                                const Style &style,
                                unsigned long helpId = 0);

virtual IMessageBox::Response show(
                                const IException &exception,
                                unsigned long helpId = 0);

virtual IMessageBox::Response show(
                                const IErrorInfo &error,
                                unsigned long helpId = 0);
```

Shows a message box. The returned value is an enumerator provided by Response (see page 527).

message The message box text.

style Use the styles provided by IMessageBox::Styles (see page 529) to specify the message box style.

helpId (Optional) If *helpId* is nonzero, a **Help** push button is displayed. When the **Help** push button is selected, the help panel with the ID corresponding to the *helpId* parameter is displayed.

severity Use the enumeration Severity (see page 527) to specify the severity of your message.

error An IErrorInfo object.

exception An IException object.

Figure 171. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Enumerations

Response

```
public:
enum Response
  {
  enter,
  ok,
  cancel,
  abort,
  retry,
  ignore,
  yes,
  no,
  unknown,
  };
```

These enumerators specify the default response for a message box. When the message box is displayed, the push button drawn with the selected emphasis is the default response.

enter
Selects the **Enter** push button.

ok
Selects the **OK** push button.

cancel
Selects the **Cancel** push button.

abort
Selects the **Abort** push button.

retry
Selects the **Retry** push button.

ignore
Selects the **Ignore** push button.

yes
Selects the **Yes** push button.

no
Selects the **No** push button.

unknown
Selects a push button that is not provided by the User Interface Class Library or the system.

Severity

```
public:
enum 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.

Chapter 118. IMessageBox::Style

Derivation

IBase
  IBitFlag
    IMessageBox::Style

Header File: imsgbox.hpp

The nested class IMessageBox::Style provides set of valid message box styles for the static member functions of the class IMessageBox (see page 525).

Styles

These styles specify message box push buttons:

abortRetryIgnoreButton
Contains **Abort**, **Retry**, and **Ignore** push buttons.

cancelButton
Contains a **Cancel** push button.

enterButton
Contains an **Enter** push button.

enterCancelButton
Contains both **Enter** and **Cancel** push buttons.

okButton
Contains an **OK** push button.

okCancelButton
Contains both **OK** and **Cancel** push buttons.

retryCancelButton
Contains both **Retry** and **Cancel** push buttons.

yesNoButton
Contains both **Yes** and **No** push buttons.

yesNoCancelButton
Contains **Yes**, **No**, and **Cancel** push buttons.

These are default selection styles:

defButton1
Specifies that the first push button be the default selection. This is the default style.

defButton2
Specifies that the second push button be the default selection.

© Copyright IBM Corp. 1994

defButton3
Specifies that the third push button be the default selection.

These styles specify icons for the message box:

errorIcon

Contains a icon.

informationIcon

Contains a icon.

noIcon
Contains no icon.

queryIcon

Contains a icon.

warningIcon

Contains a icon.

These are modality styles:

applicationModal
Specifies that the message box be application modal. This is the default style.

Note: A modal message box's owner is disabled.

systemModal
Specifies that the message box be system modal.

This is the mobility style:

moveable
Specifies that the message box be moveable.

Motif Information: 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 enterCancelButton, retryCancelButton, and abortRetryIgnoreButton 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.

Public Members

Figure 172. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 173. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 119. IMessageQueueHandle

Derivation

IBase
  IHandle
    IMessageQueueHandle

Header File: ihandle.hpp

Objects of the IMessageQueueHandle class access application message queues.

Motif Information: AIX does not support this class.

Presentation Manager Information: IMessageQueueHandle is an alias for the OS/2 Programmer's Toolkit HMQ typedef.

Constructors

```
public:
IMessageQueueHandle(Value hmq = 0);
```

You can construct objects of this class from a message queue handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IMessageQueueHandle value.

postEvent

```
static void postEvent(
              const IMessageQueueHandle &handle,
              unsigned long eventId,
              const IEventParameter1 &parm1 = 0ul,
              const IEventParameter2 &parm2 = 0ul);

void postEvent(
       unsigned long eventId,
       const IEventParameter1 &parm1 = 0ul,
       const IEventParameter2 &parm2 = 0ul) const;
```

Posts an event constructed from the parameters to the message queue handle object or other message queue handles.

© Copyright IBM Corp. 1994

postEvents

```
static void postEvents(
             unsigned long eventId,
             const IEventParameter1 &parm1 = 0ul,
             const IEventParameter2 &parm2 = 0ul);
```

Broadcasts an event to other message queue handles asynchronously.

Figure 174. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Chapter 120. IModuleHandle

Derivation

IBase
  IHandle
    IModuleHandle

Header File: ihandle.hpp

Objects of the IModuleHandle class access application modules.

Motif Information: AIX does not support this class.

Presentation Manager Information: IModuleHandle is an alias for the OS/2 Programmer's Toolkit HMODULE typedef.

Constructors

```
public:
IModuleHandle(Value hmod = 0);
```

You can construct objects of this class from a module handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IModuleHandle value.

Figure 175. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 121. IMouseClickEvent

Derivation

IBase
  IVBase
    IEvent
      IMouseClickEvent

Header File: imcevt.hpp

Objects of the IMouseClickEvent class represent a mouse click event. A mouse click handler creates an IMouseClickEvent when the user presses, releases, clicks, or double-clicks the mouse button.

Mouse click events are first dispatched to the window the mouse pointer is positioned upon. If that window does not process the mouse click 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 click event itself.

Portability Considerations: OS/2 and AIX number the buttons on a three button mouse differently. For OS/2, the middle button is button 3. For AIX, the middle button is button 2. Typically, OS/2 applications only acknowledge two mouse buttons. If an application runs on OS/2 with a three button mouse, the application uses the two outside buttons. On AIX, the application uses the first two buttons.

Motif Information: X only has two native mouse events, ButtonPress and ButtonRelease. The User Interface Class Library maps these two events to the four action enumerators of MouseAction (see page 537).

up A ButtonRelease.

down A ButtonPress.

doubleClick If a second ButtonPress event is received within the multi-click time defined by the application in X, a doubleClick is dispatched instead of a down.

click If a ButtonRelease event is received following a ButtonPress that has not been coverted to a doubleClick, a click is dispatched. Specifically, the ButtonRelease must be detected within the multi-click timeout of receiving the ButtonPress. There is no corresponding event on X.

The widgets only react to ButtonPress and ButtonRelease events. If either the down or click action result in a return value of true from the mouse click handler, the ButtonRelease event is not dispatched to the widget.

© Copyright IBM Corp. 1994

Constructors

```
public:
IMouseClickEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IMouseClickHandler::dispatchHandlerEvent (see page 540) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

handleOnTop

```
IWindowHandle handleOnTop() const;
```

Returns the handle of the window the mouse pointer is over. You can use IWindow::windowWithHandle (see page 943) to convert the return value to an IWindow*. If the window is not represented by an IWindow object, windowWithHandle returns 0.

mouseAction

```
MouseAction mouseAction() const;
```

Returns the action taken on the mouse. The enumeration MouseAction (see page 537) provides the return values.

mouseNumber

```
MouseNumber mouseNumber() const;
```

Returns the mouse button the user pressed. The enumeration MouseNumber (see page 537) provides the return values.

mousePosition

```
IPoint mousePosition() const;
```

Returns the mouse pointer position relative to the window returned by IEvent::window (see page 281).

Figure 176. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*

Figure 176. Inherited Public Members

Member function	Class
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent

Figure 176. Inherited Public Members

Member function	Class
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Enumerations

MouseAction

```
public:
enum MouseAction
  {
  click,
  doubleClick,
  down,
  up,
  };
```

These enumerators list the values IMouseClickEvent::mouseAction (see page 536) can return:

click A user pressed and released a mouse button.

doubleClick A user pressed and released a mouse button twice in succession.

down A user pressed a mouse button.

up A user released a mouse button.

MouseNumber

```
public:
enum MouseNumber
  {
  button1,
  button2,
  button3,
  button12,
```

```
};
```

These enumerators list the values IMouseClickEvent::mouseNumber (see page 536) can return:

button1 A user pressed mouse button 1.

button2 A user pressed mouse button 2.

button3 A user pressed mouse button 3.

button12 A user pressed two mouse buttons simultaneously.

Chapter 122. IMouseClickHandler

Derivation

IBase
  IVBase
    IHandler
      IMouseClickHandler

Header File: imchdr.hpp

Objects of the IMouseClickHandler class process a variety of mouse events, such as:

- Button presses and releases
- Double-clicks
- Multiple button presses

Create a handler derived from IMouseClickHandler and attach it to any window whose mouse click events you wish to handle. You can do this by calling IHandler::handleEventsFor (see page 403) to pass this window to the mouse click handler.

When the mouse click handler receives a mouse click event, it creates an object of IMouseClickEvent (see page 535) and routes that object to the virtual function IMouseClickHandler::mouseClicked (see page 540). You can override this virtual function to supply your own specialized processing of these events.

A mouse click event continues to travel up the owner chain until either a handler stops it or the mouse click is processed by the window itself. As a result, an IMouseClickHandler object should not stop the processing of any mouse click event that it does not understand since the mouse click has the potential of being handled by any window in the owner chain, such as the frame window.

The return value from the virtual function specifies whether the mouse click event is passed on for additional processing, as follows:

true The mouse click event requires no additional processing. Do not pass it to another handler.

false Pass the mouse click event to the next handler for additional processing, as follows:

- If there is another handler for the focus window, pass the mouse click event to the next handler.
- If this is the last handler for the focus window, call IWindow::defaultProcedure (see page 946) to process the mouse click event. This call might result in the event being dispatched to the focus window's owner window.

© Copyright IBM Corp. 1994

Constructors

```
public:
IMouseClickHandler();
```

Because this class has a pure virtual function, you cannot construct objects of this class.

Public Members

Figure 177. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a mouse click event is received, the appropriate virtual function is called.

mouseClicked

```
virtual Boolean mouseClicked(IMouseClickEvent &event) = 0;
```

Implemented by derived classes to handle a mouse-click event.

Figure 178. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 123. IMultiCellCanvas

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ICanvas
            IMultiCellCanvas

Header File: imcelcv.hpp

Objects of the IMultiCellCanvas class manage a collection of child windows in a way similar to that of data stored in a spread sheet. A multiple-cell canvas is a two-dimensional array of *cells* with an origin of (1,1) in the upper-left corner. Windows are added to a multiple-cell canvas by specifying a starting cell and, optionally, a number of contiguous columns, rows, or a combination thereof.

This class bases the initial cell sizes on the minimum sizes returned by each child window of the canvas. You can override the minimum sizes provided by the User Interface Class Library by doing either of the following:

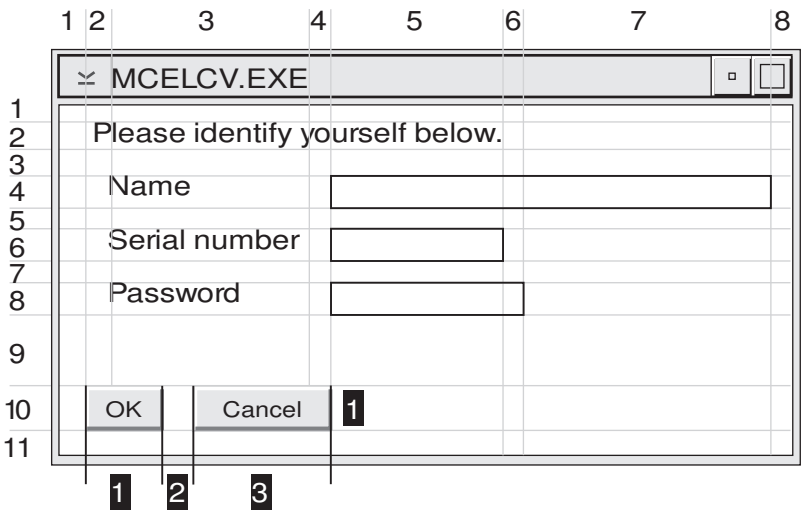
- Calling setMinimumSize
- Creating a derived class and implementing a calcMinimumSize function

The user cannot resize fixed rows or columns by changing the size of the multiple-cell canvas.

Undesirable movement of the windows might result when a multiple-cell parent canvas updates its child windows. You can avoid this by calling IStaticText::setLimit so that the static text bases its minimum size on an expected number of characters rather than its actual text string.

The following figure illustrates the positioning of three static text controls, three entry fields, and two push buttons:

© Copyright IBM Corp. 1994



Because there is no text in either row 1 or column 1, row 1 and column 1 take the default cell size. As you can see in the figure, the default cell size is relatively small and therefore preserves screen space. Also, using the default cell size for column 1 and row 1 creates some padding, or margin, around the child windows that you place in the multiple-cell canvas.

Positioning Static Text Controls Example

The first static text control has a starting cell of (2, 2). The child window that contains the static text control exists in one row, row 2, and extends from column 2 through column 7. It occupies six columns (2, 3, 4, 5, 6, and 7). The following addToCell call was used to produce it, where myClient is the client window multiple-cell canvas:

```
myClient.addToCell(&prompt, 2, 2, 6, 1);
```

The name of the child window that contains the static text control is `prompt`. The two numbers that follow the name specify the starting column and row, (2, 2). The last two numbers, (6, 1), specify that the control extends through six columns and occupies one row.

The other static text controls, which contain the text strings “Name,” “Serial number,” and “Password,” have starting cells of (3, 4), (3, 6), and (3, 8), respectively. The following addToCell calls were used to produce them:

```
myClient.addToCell(&namePrompt, 3, 4);
myClient.addToCell(&numberPrompt, 3, 6);
myClient.addToCell(&passwordPrompt, 3, 8);
```

Notice that we did not specify the number of columns and rows for these text controls to occupy. This means the columns and rows take the default, which is one column and one row. The width of column 3 is based on the static text control that contains “Serial number.”

Positioning Entry Fields Example

The first entry field has a starting cell of (5, 4) and extends through three columns (columns 5, 6, and 7) while occupying one row. The first static text control has already determined the width of these columns, so this entry field is stretched to the right edge of column 7.

The second entry field has a starting cell of (5, 6) and occupies the default of one column and one row. This means that the width of column 5 is based on the width of the second entry field, because it only occupies one column.

The third entry field has a starting cell of (5, 8), spanning two columns and one row. Here are the addToCell calls for the entry fields:

```
myClient.addToCell(&name, 5, 4, 3, 1);
myClient.addToCell(&number, 5, 6);
myClient.addToCell(&password, 5, 8, 2, 1);
```

Using Expandable Rows and Columns Example

In the figure, column 7 is much wider than you might expect. You might think that it should be the same size as all of the other columns except 3 and 5, which are sized to fit particular controls. Also, the height of row 9 is much greater than that of the other rows. The reason column 7 is so much wider and that the height of row 9 is so much greater is that we made them *expandable*. Columns and rows that are expandable grow when the window's size increases or shrink when the window's size decreases.

When the space allotted to the canvas is larger than the size needed to contain the canvas, expandable rows and columns are given more space relative to each other. For example, if one expandable row is twice the size of another expandable row, the ratio is maintained after increasing the size of the canvas. Rows and columns that are not specified as expandable maintain their fixed size.

The setColumnWidth and setRowHeight functions have an expandable parameter whose default is false. However, we set them to true for column 7 and row 9. Here is the setColumnWidth function calls:

```
myClient.setColumnWidth(7, 0, true);
```

On the setColumnWidth function call, the first number represents column 7. The second number specifies the amount that the width of the column is to increase beyond the minimum size required for the window or windows contained in the column. We set this to 0 so that the column would not take up screen space unless the canvas is wide enough that it has space not needed by the non-expandable columns. The final value, true, indicates that this column is expandable.

Positioning Push Buttons Example

The last part of this multiple-cell canvas is actually another multiple-cell canvas. The two push buttons, **OK** and **Cancel**, are contained in a multiple-cell canvas that is nested inside the multiple-cell canvas that contains the other controls. To do this, we

first defined a nested multiple-cell canvas called myButtons and added it to the bottom of the myClient multiple-cell canvas:

```
myClient.addToCell(&myButtons, 2, 10, 6, 1);
```

The myButtons canvas begins in column 2, row 10, and spans six columns of one row. Next, we used the addToCell function to put the two push buttons in the myButtons canvas, as follows:

```
myButtons.addToCell(&ok, 1, 1);
myButtons.addToCell(&cancel, 3, 1);
```

The **OK** push button begins in column 1, row 1 of the nested canvas and occupies one column. In the figure, the white numbers in black boxes indicate the columns and rows in the nested canvas. The **Cancel** push button begins in column 3, row 1 of the nested canvas and also spans one column. The width of the columns is determined by the size of the text in the push buttons, together with the margin around the text (the space between the text and the border of the push buttons).

You must specify the last row and column of a multiple-cell canvas on either an addToCell call or on setRowHeight and setColumnWidth calls. This is done to provide a padding, or margin, for the right and bottom sides of the canvas, similar to the margin provided by taking the default cell size for column 1 and row 1. The following are the calls used in the example to set the last column and row, column 8 and row 11, to the default column width and row height:

```
myClient.setColumnWidth(8, IMultiCellCanvas::defaultCell().width());
myClient.setRowHeight(11, IMultiCellCanvas::defaultCell().height());
```

The following points are apparent from the preceding example:

- It is not necessary to have a window in every cell.
- You can use empty columns and empty rows to provide padding around the windows.
- The canvas can have more rows or columns than needed to hold the additional windows. For example, columns 1 and 8, as well as rows 1, 3, and 5, are empty. They will be sized either to a default value or, they can be sized explicitly by the application.
- Windows can occupy multiple columns, rows, or both. The first entry field, for example, begins in cell (5, 4) and occupies columns 5 through 7. The number of columns or rows occupied includes the originating column or row.
- A window's range can overlap cells that contain other windows. However, two windows cannot have the same starting cell.

Nested Classes

IMultiCellCanvas::Style

Constructors

```
public:
IMultiCellCanvas(
   unsigned long windowIdentifier,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initialSize = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

You can construct objects of this class using the following parameters:

windowIdentifier
Window identifier of the canvas you construct.

Note: For OS/2, this value is limited to the range 0 through 65535.

It is recommended that you:

- Give unique identifiers to all windows in the same frame window. *In the same window* means that the frame window is the first frame in their parent window chain.
- Give the client window a window ID of FID_CLIENT (0x8008).
- Do not use the other FID_xxx values.

parent The parent window of the canvas you construct. You must specify a parent window. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0. The parent window is primarily used for visible relationships.

owner The owner window of the canvas you construct.

Note: In OS/2, the owner window is primarily used for routing notification events and unprocessed messages. OS/2 also uses the owner window chain to inherit colors.

In Motif, the owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initialSize (Optional) The initial position and size of the canvas you construct. The initial position is the lower-left corner of the canvas relative to the lower-left corner of the parent window.

style (Optional) Use the styles provided by IMultiCellCanvas::Style (see page 555) to specify the control's styles.

Public Members

addToCell

```
virtual IMultiCellCanvas &addToCell(
                           IWindow *childWindow,
                           unsigned long startingColumn,
                           unsigned long startingRow,
                           unsigned long numberOfColumns = 1,
```

```
                                unsigned long numberOfRows = 1);
```

Specifies the starting cell into which a window is placed. Optionally, you can specify the number of columns or rows that the window occupies. By default, a multiple-cell canvas has no windows. Therefore, you must call this function to add windows to the canvas.

Note: If the multiple-cell canvas is already shown, this function does not cause it to update the canvas itself immediately. You must call IWindow::refresh (see page 935) to update the appearance of the canvas.

You do not specify how many rows and columns a multiple-cell contains. The User Interface Class Library derives the maximum number of rows and columns from addToCell, setColumnWidth, and setRowHeight.

childWindow
The child window that you are assigning to a cell of the canvas. If this window is not a child window of the canvas, the User Interface Class Library throws an IInvalidParameter exception when the multiple-cell canvas runs its layout routine.

startingColumn
The left-most column where the child window is placed. Columns in a multiple-cell canvas are numbered beginning with 1.

startingRow
The top-most row where the child window is placed. Rows in a multiple-cell canvas are numbered beginning with 1.

numberOfColumns
(Optional) Number of columns that the child window will span. The default value is 1.

numberOfRows
(Optional) Number of rows that the child window will span. The default value is 1.

columnWidth

```
unsigned long columnWidth(unsigned long column) const;
```

Returns the current width of the specified column, in pixels.

column Identifier of the column. Columns in a multiple-cell canvas are numbered beginning with 1.

defaultCell

```
static ISize defaultCell();
```

Queries the current default cell size. The initial default size is 10 pixels by 10 pixels. You can change it with IMultiCellCanvas::setDefaultCell.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 555) unless you have changed the style using setDefaultStyle (see page 550).

disableDragLines

Condition: (1) Ignored on AIX

```
virtual IMultiCellCanvas &disableDragLines();
```

Removes the style dragLines (see page 555) from a multiple-cell canvas.

disableGridLines

Condition: (1) Ignored on AIX

```
virtual IMultiCellCanvas &disableGridLines();
```

Removes the style gridLines (see page 555) from a multiple-cell canvas.

enableDragLines

Condition: (1) Ignored on AIX

```
virtual IMultiCellCanvas &enableDragLines(
                            Boolean enable = true);
```

Sets the style dragLines (see page 555) for a multiple-cell canvas.

enableGridLines

Condition: (1) Ignored on AIX

```
virtual IMultiCellCanvas &enableGridLines(
                            Boolean enable = true);
```

Sets the style gridLines (see page 555) for a multiple-cell canvas.

hasDragLines

Condition: (1) Ignored on AIX

```
Boolean hasDragLines() const;
```

Queries whether a multiple-cell canvas has the style dragLines (see page 555) set.

hasGridLines

Condition: (1) Ignored on AIX

```
Boolean hasGridLines() const;
```

Queries whether a multiple-cell canvas has the style gridLines (see page 555) set.

isColumnExpandable

```
Boolean isColumnExpandable(unsigned long column) const;
```

If the specified column can be expanded when the canvas is larger than its minimum size, true is returned. You can use setColumnWidth to make a column expandable.

column Identifier for the column. Columns in a multiple-cell canvas are numbered beginning with 1.

isRowExpandable

```
Boolean isRowExpandable(unsigned long row) const;
```

If the specified row can be expanded when the canvas is larger than its minimum size, true is returned. You can use setRowHeight to make a row expandable.

row Identifier for the row. Rows in a multiple-cell canvas are numbered beginning with 1.

removeFromCell

```
virtual IWindow *removeFromCell(IWindow *childWindow);

virtual IWindow *removeFromCell(
                    unsigned long column,
                    unsigned long row);
```

Removes a window from the canvas. If you specify a cell, the window that occupies that cell is removed. If you specify a window, the window is removed.

This function returns the window removed from the canvas. It does not delete the child window. If the specified window is not found, 0 is returned.

You must call this function if you delete the child window or change its parent window.

Note: If the multiple-cell canvas is already shown, this function does not cause it to update immediately. You must call IWindow::refresh (see page 935) to update the appearance of the canvas.

childWindow
The child window to remove from the multiple-cell canvas.

column The starting column of the child window being removed. Columns are numbered beginning with 1. This is the starting column you specified in the addToCell call that added the child window to the canvas.

row The starting row of the child window being removed. Rows are numbered beginning with 1. This is the starting row you specified in the addToCell call that added the child window to the canvas.

rowHeight

```
unsigned long rowHeight(unsigned long row) const;
```

Returns the current height of the specified row, in pixels.

row Identifier for the row. Rows in a multiple-cell canvas are numbered beginning with 1.

setColumnWidth

```
virtual IMultiCellCanvas &setColumnWidth(
                            unsigned long column,
                            unsigned long widthInPixels,
                            Boolean expandable = false);
```

Defines the smallest width that the column will have. The minimum sizes of the child windows in this column might cause the actual width of this column to be larger than this value.

If you do not call this function, the column will have a minimum width of 0, unless the column contains no child windows. In that case, the canvas sizes the column to the default cell width.

You do not specify how many rows and columns a multiple-cell canvas contains. The User Interface Class Library derives the maximum number of rows and columns from addToCell, setColumnWidth, and setRowHeight.

Note: If the multiple-cell canvas is already shown, this function does not cause it to update immediately. You must call IWindow::refresh (see page 935) to update the appearance of the canvas.

column Identifier for the column. Columns in a multiple-cell canvas are numbered beginning with 1.

widthInPixels
Minimum width in pixels.

expandable
(Optional) If set to true, windows in the column are expanded horizontally when the canvas is larger than its minimum size. The default is false.

setDefaultCell

```
static void setDefaultCell(const ISize &widthAndHeight);
```

Sets the default width for a column and height for a row of a cell that does not have a child window. The initial default size for both rows and columns is 10 pixels.

Note: This function does not cause the layout of a multiple-cell canvas to update the canvas itself immediately. To force a multiple-cell canvas to update after calling this function, call setLayoutDistorted (see page 550), as follows:

```
setLayoutDistorted(IWindow::layoutChanged |
                        IWindow::immediateUpdate, 0);
```

setDefaultStyle

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

Sets the default style for all subsequent canvases.

style Use the styles provided by IMultiCellCanvas::Styles (see page 555) to specify the default style.

setLayoutDistorted

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

Adds the IWindow::layoutDistorted flag to its internal state if it receives a value of IWindow::childWindowCreated or IWindow::childWindowDestroyed and calls IWindow::setLayoutDistorted. If the IWindow::immediateUpdate flag is on, the canvas runs its layout routine.

The internal state of a window is indicated by the Layout (see page 953) enumeration.

A window treats the following events as changes to the layout of its child windows:

- Changing the size
- Changing the size of a child window
- Adding another child window
- Deleting a child window

setRowHeight

```
virtual IMultiCellCanvas &setRowHeight(
                          unsigned long row,
                          unsigned long heightInPixels,
                          Boolean expandable = false);
```

Defines the smallest height that the row will have. The minimum sizes of the child windows in this row can cause the actual height of the row to be larger than this value.

If you do not call this function, the row will have a minimum height of 0, unless the row contains no child windows. In that case, the canvas sizes the row to the default cell height.

You do not specify how many rows and columns a multiple-cell canvas contains. The User Interface Class Library derives the maximum number of rows and columns from addToCell, setColumnWidth, and setRowHeight.

Note: If a multiple-cell canvas is already shown, this function does not cause it to update the canvas itself immediately. You must call IWindow::refresh (see page 935) to update the appearance of the canvas.

row Identifier for the row. Rows in a multiple-cell canvas are numbered beginning with 1.

heightInPixels
Minimum height in pixels.

expandable
(Optional) If set to true, windows in the row are expanded vertically when the canvas is larger than its minimum size. The default is false.

windowInCell

```
virtual IWindow *windowInCell(
                  unsigned long startingColumn,
                  unsigned long startingRow) const;
```

Returns the child window occupying the specified cell.

startingColumn
Identifier for the column. Columns in a multiple-cell canvas are numbered beginning with 1.

startingRow
Identifier for the row. Rows in a multiple-cell canvas are numbered beginning with 1.

Figure 179. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	ICanvas

Figure 179. Inherited Public Members

Member function	Class
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
defaultPushButton	ICanvas
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl

Figure 179. Inherited Public Members

Member function	Class
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isTabStop	ICanvas
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow

Figure 179. Inherited Public Members

Member function	Class
matchForMnemonic	ICanvas
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
origDefaultButtonHandle	ICanvas
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow

Figure 179. Inherited Public Members

Member function	Class
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ICanvas
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

layout

```
virtual IMultiCellCanvas &layout();
```

Computes the child windows' positions and sizes based on:

- The contents of the cells
- The minimum size of rows and columns
- The expandability of rows and columns

Figure 180. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
areChildrenReversed	ICanvas
calcMinimumSize	IWindow

Figure 180. Inherited Protected Members

Member function	Class
calcMinimumSize	ICanvas
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow

Figure 180. Inherited Protected Members

Member function	Class
fixupChildren	ICanvas
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
layout	ICanvas
layoutSize	ICanvas
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ICanvas
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	ICanvas
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setLayoutSize	ICanvas
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	ICanvas
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 124. IMultiCellCanvas::Style

Derivation

IBase
  IBitFlag
    IMultiCellCanvas::Style

Header File: imcelcv.hpp

The nested class IMultiCellCanvas::Style provides a set of valid multiple-cell canvas styles for defaultStyle, setDefaultStyle, and the constructor of the IMultiCellCanvas (see page 541) class. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (see page 70)
IWindow::Style (see page 957)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

dragLines
Causes the multiple-cell canvas to have draggable grid lines between the rows and columns of the canvas.

gridLines
Causes the multiple-cell canvas to have grid lines between the rows and columns of the canvas.

Motif Information: dragLines and gridLines are not supported in the AIX release.

Public Members

Figure 181. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 182. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 125. IMultiLineEdit

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IMultiLineEdit
```

Header File: imle.hpp

Objects of the IMultiLineEdit class create and manage multiple-line edit (MLE) controls, also known as text widgets. 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 (see page 247)
- IEditVerifyHandler (see page 253)
- IFocusHandler (see page 307)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)
- IScrollHandler (see page 760)

The MLE control can enable screen refreshes during its member functions processing. In doing so, it negates any previous use of IMultiLineEdit::disableRefresh.

Note: Do not load text containing nulls or any other imbedded non-printable characters into an IMultiLineEdit object. Because the user cannot see these characters, the user is likely to type over, delete, or otherwise destroy these characters.

Portability Considerations: The Motif XmText widget does not support imbedded NULLs or imbedded non-printable characters. Only the text preceding the first NULL is loaded into the control.

For a portable application, do not load text containing imbedded NULLs or non-printable characters into an IMultiLineEdit object. Additionally, remember end-of-line characters are different on different systems.

Motif Information: The User Interface Class Library implements this class with the XmText widget. The XmText widget does not support imbedded NULLs or imbedded non-printable 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 (see page 576), the member functions ignore it and use noTran instead.

© Copyright IBM Corp. 1994

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 proportional font (to ensure good performance). This means that, if 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.

Using tab characters can also confuse line-oriented functions. Typically, Motif assumes tabs are 10 blanks, but if a line ends with a tab it might cause an incorrect line count.

Presentation Manager Information: 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.

Nested Classes

IMultiLineEdit::Style

Constructors

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

public:
IMultiLineEdit(unsigned long id, IWindow *parent);

public:
IMultiLineEdit(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Using the parent window, owner window, optional size and location, and optional style parameters
- Using the parent window
- Using the handle of an existing MLE

id An MLE control ID.

parent The parent window.

owner The owner window.

initial (Optional) A rectangle for the control. It specifies the initial position and size of the object you are constructing.

style (Optional) The initial style for the control. The default is classDefaultStyle (see page 577).

handle The window handle of an existing MLE control.

Public Members

add

```
virtual IMultiLineEdit &add(
                          const char *text,
                          unsigned long textSize = 0,
                          EOLFormat type = cfText);
```

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 \r\n characters. Therefore, if you need to explicitly use \r\n, do one of the following:

- Use \r\n at the beginning of an input string
- Use IMultiLineEdit::addLine (see page 560)

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 (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

addAsLast

```
virtual IMultiLineEdit &addAsLast(
                          const char *text,
                          unsigned long textSize = 0,
                          EOLFormat type = cfText);
```

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 (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

addAtOffset

```
virtual IMultiLineEdit &addAtOffset(
                        const char *text,
                        unsigned long charnumber,
                        unsigned long textSize = 0,
                        EOLFormat type = cfText);
```

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 (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

addLine

```
virtual IMultiLineEdit &addLine(
                        const char *text,
                        unsigned long lineNumber,
                        EOLFormat type = cfText);
```

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 (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded 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

```
virtual IMultiLineEdit &addLineAsLast(
                          const char *text,
                          EOLFormat type = cfText);
```

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 (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded 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.

backgroundColor

Condition: (2) OS/2 Only

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

Returns the background color value of the MLE or the default if no color for the area has been set.

clear

```
virtual IMultiLineEdit &clear(unsigned long timestamp = 0);
```

Replaces the currently selected text with blank spaces. No exception is thrown if there is no text selected.

Because this function is system-independent, you can either call clear() or clear(1100) on any system.

Motif Information: The optional parameter *timestamp* is provided to allow a timestamp value to be passed when this function is called from an event handler.

In AIX, you can get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Presentation Manager Information: The OS/2 release does not use the *timestamp* parameter.

copy

```
virtual IMultiLineEdit &copy(unsigned long timestamp = 0);
```

Copies the currently selected text from the MLE to the clipboard. No exception is thrown if there is no text selected.

Because this function is system-independent, you can either call copy() or copy(1100) on any system.

Motif Information: The optional parameter *timestamp* is provided to allow a timestamp value to be passed when this function is called from an event handler.

In AIX, you can get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Presentation Manager Information: The OS/2 release does not use the *timestamp* parameter.

cursor

```
unsigned long cursor() const;
```

Returns the line number of the line that contains the cursor.

cut

```
virtual IMultiLineEdit &cut(unsigned long timestamp = 0);
```

Copies the currently selected text from the MLE to the clipboard and then deletes the selected text from the MLE. No exception is thrown if there is no text selected.

Because this function is system-independent, you can either call cut() or cut(1100) on any system.

Motif Information: The optional parameter *timestamp* is provided to allow a timestamp value to be passed when this function is called from an event handler.

In AIX, you can get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Presentation Manager Information: The OS/2 release does not use the *timestamp* parameter.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 577) unless you have changed the style using setDefaultStyle (see page 570).

disableDataUpdate

```
virtual IMultiLineEdit &disableDataUpdate();
```

Prevents changes to the current MLE text.

disableRefresh

```
virtual IMultiLineEdit &disableRefresh();
```

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.

Note: This function is obsolete. Use IMultiLineEdit::disableUpdate instead.

Motif Information: Enabling and disabling screen refreshes is not typically done in Motif, as screen refresh management is handled automatically by X. However, to ensure the portability of logic which assumes that disabling refresh also disables input temporarily, the User Interface Class Library disables input in Motif also. Screen refresh is always enabled.

disableUpdate

```
virtual IMultiLineEdit &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.

Motif Information: Enabling and disabling screen refreshes is not typically done in Motif, as screen refresh management is handled automatically by X. However, to ensure the portability of logic which assumes that disabling refresh also disables input temporarily, the User Interface Class Library disables input in Motif also. Screen refresh is always enabled.

disableWordWrap

```
virtual IMultiLineEdit &disableWordWrap();
```

Disables word-wrap mode in the MLE.

discard

```
virtual IMultiLineEdit &discard();
```

Deletes all currently selected text. No exception is thrown if there is no text selected.

editRegionHeight

```
unsigned long editRegionHeight() const;
```

Returns the height of the edit region.

editRegionWidth

```
unsigned long editRegionWidth() const;
```

Returns the width of the edit region.

enableDataUpdate

```
virtual IMultiLineEdit &enableDataUpdate(
                          Boolean update = true);
```

Sets the MLE to read-write mode, which lets the user update the MLE text.

enableNotification

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

Causes the MLE control to send notifications to any observer objects added.

enable The default of true sets notification on.

enableRefresh

```
virtual IMultiLineEdit &enableRefresh(
                          Boolean refresh = true);
```

Enables screen updates.

Note: This function is obsolete. Use IMultiLineEdit::enableUpdate instead.

Motif Information: Enabling and disabling screen updates is not typically done in Motif, as screen refresh management is handled automatically by X. However, to ensure the portability of logic which assumes that disabling update also disables input temporarily, the User Interface Class Library disables input in Motif also.

enableUpdate

```
virtual IMultiLineEdit &enableUpdate(Boolean update = true);
```

Enables screen updates.

Motif Information: Enabling and disabling screen updates is not typically done in Motif, as screen refresh management is handled automatically by X. However, to ensure the portability of logic which assumes that disabling update also disables input temporarily, the User Interface Class Library disables input in Motif also.

enableWordWrap

```
virtual IMultiLineEdit &enableWordWrap(
                          Boolean enable = true);
```

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 MLEs window. The text is still one line of text. No carriage control or line feed characters are added.

exportSelectedTextToFile

```
virtual unsigned long exportSelectedTextToFile(
                       const char *fileName,
                       EOLFormat type = cfText);
```

Saves the currently marked text to the specified file and returns the number of bytes written to the file. No exception is thrown if there is no text selected.

fileName The name of the file you want the text saved to.

type Use the enumeration EOLFormat (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX only supports the noTran format enumerator.

exportToFile

```
virtual unsigned long exportToFile(
                       const char *fileName,
                       EOLFormat type = cfText);
```

Saves the contents of the MLE to the specified file and returns the number of bytes written to the file.

Note: You cannot save files larger than 64 Kb. This function saves the first 64 Kb.

fileName The name of the file you want the text saved to.

type Use the enumeration EOLFormat (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX only supports the noTran format enumerator.

foregroundColor

Condition: (2) OS/2 Only

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

Returns the foreground color value of the MLE or the default if no color for the area has been set.

hasSelectedText

```
virtual Boolean hasSelectedText() const;
```

Queries whether any of the MLE's text is selected.

importFromFile

```
virtual unsigned long importFromFile(
                      const char *fileName,
                      EOLFormat type = cfText);
```

Inserts the contents of the specified file into the MLE at the current cursor position and returns the number of bytes imported.

Note: You cannot import files larger than 64 Kb. This function imports the first 64 Kb.

fileName The name of the file you want the text imported from.

type Use the enumeration EOLFormat (see page 576) to specify the type of format. The default is cfText.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

isChanged

```
Boolean isChanged() const;
```

Queries whether any changes have been made to the MLE since the last time the changed flag was reset.

isDataUpdateEnabled

```
Boolean isDataUpdateEnabled() const;
```

Queries whether the MLE is in read-only mode.

isReadOnly

```
Boolean isReadOnly() const;
```

Queries whether the MLE is in read-only mode.

Note: This function is obsolete. Use IMultiLineEdit::isDataUpdateEnable instead.

isUndoable

Condition: (1) Ignored on AIX

```
Boolean isUndoable() const;
```

Queries whether any undoable actions have been performed on the contents of the MLE.

Motif Information: This function always returns false.

isWordWrap

```
Boolean isWordWrap() const;
```

Queries whether the MLE is in word-wrap mode.

limit

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

Returns the currently set number of bytes the MLE can hold.

numberOfLines

```
unsigned long numberOfLines() const;
```

Returns the number of displayed lines, 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.

Motif Information: If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

paste

```
virtual IMultiLineEdit &paste();
```

Inserts the contents of the clipboard into the MLE at the cursor position. No exception is thrown if there is no text selected.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL. AIX only supports the noTran format enumerator.

removeAll
```
virtual IMultiLineEdit &removeAll();
```

Deletes the entire contents of the MLE.

removeChangedFlag
```
virtual IMultiLineEdit &removeChangedFlag();
```

Resets the changed flag so that from this point forward changes to the MLE can be detected.

Note: This function is obsolete. Use IMultiLineEdit::resetChangedFlag instead.

removeLine
```
virtual IMultiLineEdit &removeLine(
                          unsigned long lineNumber);
```

Deletes the specified line of text.

Motif Information: If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

resetChangedFlag
```
virtual IMultiLineEdit &resetChangedFlag();
```

Resets the changed flag so that from this point forward changes to the MLE can be detected.

selectedRange
```
IRange selectedRange() const;
```

Returns the range of the selected text. If no text is selected, a range of (cursor position, cursor position) is returned. The selected range is the index of the first character selected and the index of the last character selected. The index is 0-based.

selectedText
```
virtual IString selectedText() const;
```

Returns the selected text string. No exception is thrown if there is no text selected; a null IString is returned.

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

Returns the size of the selected area, in bytes. The length includes CRLFs, but not NULL terminators. No exception is thrown if there is no text selected; 0 is returned.

selectRange

```
virtual IMultiLineEdit &selectRange(
                          constIRange&range = IRange(0,end),
                          unsigned long timestamp = 0);
```

Selects a range of text. If you specify a range where the lower and upper values are equal, the selection is an insertion point.

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.

timestamp In AIX, a value within the time member of the XEvent structure.

Motif Information: The optional parameter *timestamp* is provided to allow a timestamp value to be passed when this function is called from an event handler.

In AIX, you can get the value for *timestamp* from the time member of the XEvent structure. The default value is interpreted as CurrentTime.

Presentation Manager Information: The OS/2 release does not use the *timestamp* parameter.

setChangedFlag

```
virtual IMultiLineEdit &setChangedFlag(
                          Boolean changed = true);
```

Sets a flag to indicate the MLE contents have changed.

setCursorAt

```
virtual IMultiLineEdit &setCursorAt(
                          unsigned long cursorPosition);
```

Moves the cursor a specified number of positions from left to right in the MLE. The count begins at the first position in the MLE, not at the cursor's current position.

setCursorAtLine

```
virtual IMultiLineEdit &setCursorAtLine(
                          unsigned long lineNumber);
```

Moves the cursor position to the first position of the specified line.

Motif Information: If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

setDefaultStyle

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

Sets the default style for all subsequent MLEs.

style Use the styles provided by IMultiLineEdit::Styles (see page 577) to specify the default style.

setEditRegion

```
virtual IMultiLineEdit &setEditRegion(
                          const ISize &sizeEditRegion);

virtual IMultiLineEdit &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.

Motif Information: In Motif, if the window is too small to show all the text, it is not an error. In this situation, this function does not throw an exception.

setEditRegionHeight

```
virtual IMultiLineEdit &setEditRegionHeight(long height);
```

Sets the height of the edit region.

Motif Information: In Motif, if the window is too small to show all the text, it is not an error. In this situation, this function does not throw an exception.

setEditRegionWidth

```
virtual IMultiLineEdit &setEditRegionWidth(long width);
```

Sets the width of the edit region.

Motif Information: In Motif, if the window is too small to show all the text, it is not an error. In this situation, this function does not throw an exception.

setFont

Condition: (2) OS/2 Only

```
virtual IMultiLineEdit &setFont(const IFont &font);
```

Changes the font of the displayed text.

setLimit

```
virtual IMultiLineEdit &setLimit(unsigned long newLimit);
```

Sets a limit on the number of bytes that can be input into the MLE. If the MLE already contains more bytes than the proposed limit, an exception of type IAccessError is thrown.

setTab

Condition: (1) Ignored on AIX

```
virtual IMultiLineEdit &setTab(
                          unsigned long tabPixelInterval);
```

Sets the number of pixels between tab stops inside the MLE.

setText

```
virtual IMultiLineEdit &setText(const char *text);

virtual IMultiLineEdit &setText(const IResourceId &text);

virtual IMultiLineEdit &setText(
                          const char *buffer,
                          unsigned long bufferSize);
```

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.

Motif Information: AIX does not support imbedded NULLs. This function only loads the text up to the first NULL.

setTop

```
virtual IMultiLineEdit &setTop(unsigned long lineNumber);
```

Makes the specified line the top-most visible line on the screen.

text

```
virtual IString text() const;

virtual IString text(unsigned long lineNumber) const;
```

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.

textLength

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

Returns the current length of the MLE text, in bytes. The length includes any CRLFs present.

top

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

Returns the line number of the line currently visible at the top of the screen.

topHandle

Condition: (3) AIX only

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

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.

undo

Condition: (1) Ignored on AIX

```
virtual IMultiLineEdit &undo();
```

Restores the MLE contents to the state they were in before the last change. If the undo fails, the change flag is reset.

visibleLines

```
unsigned long visibleLines() const;
```

Returns the number of lines that completely fit inside the edit region of the MLE. This value is based on the current font.

Motif Information: If you use a proportional font in the control, line numbers cannot be calculated accurately. Use character-oriented functions instead.

Figure 183. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow

Figure 183. Inherited Public Members

Member function	Class
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl

Figure 183. Inherited Public Members

Member function	Class
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow

Figure 183. Inherited Public Members

Member function	Class
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow

Figure 183. Inherited Public Members

Member function	Class
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

passEventToOwner

Condition: (3) AIX only

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

Determines if the event is passed on to the owner.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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 the registerCallbacks member function, the override must call Inherited::registerCallbacks to register the applicable callbacks.

This class does not add any X event handlers.

unregisterCallbacks

Condition: (3) AIX only

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

Removes any callbacks and X event handlers added by the registerCallbacks function.

Figure 184. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow

Figure 184. Inherited Protected Members

Member function	Class
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

EOLFormat

```
public:
enum 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 CRLF as denoting hard line-breaks, and ignores the sequence CRCRLF. On export, uses CRLF to denote a hard line-break and CRCRLF to denote a soft line-break caused by word-wrapping.

Motif Information: AIX only supports the noTran format enumerator. If you specify any other EOLFormat (see page 576), the member functions ignore it and use noTran instead.

Chapter 126. IMultiLineEdit::Style

Derivation

IBase
  IBitFlag
    IMultiLineEdit::Style

Header File: imle.hpp

The nested class IMultiLineEdit::Style provides a set of valid MLE styles for the member functions of the class IMultiLineEdit (see page 557). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

border
Draws a thin border around the MLE.

classDefaultStyle
Specifies the original default style for this class, which is border | vertScroll | wordWrap | IWindow::visible.

horizontalScroll
Adds a horizontal scroll bar to the MLE.

ignoreTab
Causes the MLE to ignore tab key strokes. Member functions send the keyboard events to the owner of the MLE instead.

readOnly
Restricts the user from entering any data.

verticalScroll
Adds a vertical scroll bar to the MLE.

wordWrap
Causes text to word-wrap at the end of a line.

Public Members

Figure 185. Inherited Public Members

Member function	Class
asDebugInfo	IBase*

Figure 185. Inherited Public Members

Member function	Class
asString	IBase*

© Copyright IBM Corp. 1994

Figure 185. Inherited Public Members

Member function	Class
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 186. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 127. INotebook

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          INotebook
```

Header File: inotebk.hpp

Objects of the INotebook class create and manage a notebook control.

A notebook control contains:

- A notebook client area
- A binding area
- Optional tabs
- An optional status area

When working with INotebook objects, you can also use the following nested classes:

- INotebook::PageSettings (see page 600), whose objects define the characteristics of a page when it is added to a notebook
- INotebook::Cursor (see page 597), whose objects traverse the pages of the notebook

Nested Classes

INotebook::Cursor
INotebook::PageSettings
INotebook::Style

Constructors

```
public:
INotebook(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

Condition: (2) OS/2 Only

```
public:
INotebook(unsigned long windowId, IWindow *parentAndOwner);
```

Condition: (2) OS/2 Only

© Copyright IBM Corp. 1994

```
public:
INotebook(const IWindowHandle &handle);
```

You can create a notebook control object using the following:

- The parent window, owner window, and optional size and location, and optional style parameters.
- The parent window.
- The handle of an existing notebook control.

windowId A notebook control ID.

parent The parent window.

owner The owner window.

initial (Optional) A rectangle for the notebook control. It specifies the initial position and size of the notebook you are constructing. The default constructor is IRectangle (see page 717).

style (Optional) The initial style for the notebook control. The default is classDefaultStyle

windowId The ID of a notebook control on a dialog window.

parentAndOwner
The parent window.

handle A window handle of an existing notebook control.

Public Members

addFirstPage

```
virtual IPageHandle addFirstPage(
                    const PageSettings &pageInfo,
                    IWindow *window = 0);
```

Adds a window as the first page in the notebook using specified settings. If the window pointer is null, a page with the indicated settings, but no window, is added to the notebook.

Note: When adding an IFrameWindow (see page 352) as a notebook page, create an IFrameWindow with the notebook as its parent.

pageInfo The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

window The window for the notebook page.

addLastPage

```
virtual IPageHandle addLastPage(
                      const PageSettings &pageInfo,
                      IWindow *window = 0);
```

Adds a window as the last page in the notebook using specified settings. If the window pointer is null, a page with the indicated settings, but no window, is added to the notebook.

Note: When adding an IFrameWindow (see page 352) as a notebook page, create an IFrameWindow with the notebook as its parent.

pageInfo The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

window The window for the notebook page.

addPageAfter

```
virtual IPageHandle addPageAfter(
                      const PageSettings &pageInfo,
                      const Cursor &cursor,
                      IWindow *window = 0);

virtual IPageHandle addPageAfter(
                      const PageSettings &pageToAdd,
                      const IPageHandle &referencePage,
                      IWindow *window = 0);
```

Adds a window to the notebook following the specified page using specified settings. If the window pointer is NULL, a page with the indicated settings, but no window, is added to the notebook.

Note: When adding an IFrameWindow (see page 352) as a notebook page, create an IFrameWindow with the notebook as its parent.

pageInfo The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

cursor A cursor object.

window The window for the notebook page.

pageToAdd
The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

referencePage
The page behind which you want to add the new notebook page.

addPageBefore

```
virtual IPageHandle addPageBefore(
                      const PageSettings &pageInfoToAdd,
                      const IPageHandle &referencePage,
                      IWindow *window = 0);
```

```
virtual IPageHandle addPageBefore(
                     const PageSettings &pageInfo,
                     const Cursor &cursor,
                     IWindow *window = 0);
```

Adds a window to the notebook before the specified page using specified settings. If the window pointer is NULL, a page with the indicated settings, but no window, is added to the notebook.

Note: When adding an IFrameWindow (see page 352) as a notebook page, create an IFrameWindow with the notebook as its parent.

pageInfo The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

cursor A cursor object.

window The window for the notebook page.

pageInfoToAdd
The INotebook::PageSettings (see page 600) object in which you specify the new page's settings.

referencePage
The page in front of which you want to add the new notebook page.

backgroundColor

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

Returns the background color value of the notebook area, or the default if no color for the area has been set.

backgroundMajorColor

```
virtual IColor backgroundMajorColor() const;
```

Returns the major tab background color value of the notebook, or the default if no color for the area has been set.

backgroundMinorColor

```
virtual IColor backgroundMinorColor() const;
```

Returns the minor tab background color value of the notebook, or the default if no color for the area has been set.

backgroundPageColor

```
virtual IColor backgroundPageColor() const;
```

Returns the background page color value of the notebook area, or the default if no color for the area has been set.

binding
```
virtual Binding binding() const;
```

Returns the type of binding used for the notebook. The returned value is an enumerator provided by Binding (see page 595).

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

Returns the default style. The default style is classDefaultStyle (see page 607) unless you have changed the style using setDefaultStyle (see page 587).

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

Causes the window to send notifications to any observer objects.

firstPage
```
virtual IPageHandle firstPage() const;
```

Returns an IPageHandle (see page 634) object that references the first page in the notebook.

foregroundMajorColor
```
virtual IColor foregroundMajorColor() const;
```

Returns the major tab foreground color value of the notebook, or the default if no color for the area has been set.

foregroundMinorColor
```
virtual IColor foregroundMinorColor() const;
```

Returns the minor tab foreground color value of the notebook, or the default if no color for the area has been set.

hiliteBackgroundColor
Condition: (1) Ignored on AIX

```
virtual IColor hiliteBackgroundColor() const;
```

Returns the highlight background color value of the notebook area, or the default if no color for the area has been set.

isEmpty

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

Queries whether the notebook has any pages.

lastPage

```
virtual IPageHandle lastPage() const;
```

Returns an IPageHandle (see page 634) object that references the last page in the notebook.

nextPage

```
virtual IPageHandle nextPage(
                    const IPageHandle &referencePage) const;
```

Returns an IPageHandle (see page 634) object that references the page following the specified IPageHandle in the notebook.

referencePage
The page preceding the page whose handle you want.

notebookSize

```
virtual ISize notebookSize(const IPageHandle &page) const;
```

Returns the size that the notebook must be in order to contain the specified page.

page The page that the returned size is based on.

orientation

```
virtual Orientation orientation() const;
```

Returns the current orientation of the notebook, which is the location of the major tabs and the back pages. The returned value is an enumerator provided by Orientation (see page 595).

pageSettings

```
PageSettings pageSettings(const Cursor &cursor) const;

PageSettings pageSettings(const IPageHandle &page) const;
```

Returns information about the specified page.

page Reference to the page of the notebook.

cursor A notebook page cursor object.

pageSize

```
virtual ISize pageSize() const;
```

Returns the page's size. All notebook pages have the same size. Windows associated with the notebook pages use this notebook control.

pagesToEnd

```
virtual unsigned long pagesToEnd(
                      const IPageHandle &page) const;

virtual unsigned long pagesToEnd(
                      const Cursor &cursor) const;
```

Returns the number of pages in the notebook from a specified page to the end of the notebook. This value includes the specified page.

page Reference to the page of the notebook.

cursor A notebook page cursor object.

pagesToMajorTab

```
virtual unsigned long pagesToMajorTab(
                      const Cursor &cursor) const;

virtual unsigned long pagesToMajorTab(
                      const IPageHandle &page) const;
```

Returns the number of pages in the notebook between a specified page and the next page that has a major tab.

pagesToMinorTab

```
virtual unsigned long pagesToMinorTab(
                      const Cursor &cursor) const;

virtual unsigned long pagesToMinorTab(
                      const IPageHandle &page) const;
```

Returns the number of pages in the notebook between a specified page and the next page that has a minor tab.

previousPage

```
virtual IPageHandle previousPage(
                    const IPageHandle &referencePage) const;
```

Returns an IPageHandle (see page 634) object that references the page before the specified IPageHandle.

refreshTabs

```
virtual INotebook &refreshTabs();
```

Causes the visible notebook's tabs to be repainted.

removeAllPages

```
virtual INotebook &removeAllPages();
```

Removes all the pages in the notebook.

removePage

```
virtual INotebook &removePage(const IPageHandle &page);

virtual INotebook &removePage(const Cursor &cursor);
```

Removes the specified page from the notebook.

removeTabSection

```
virtual INotebook &removeTabSection(
                     const IPageHandle &page);

virtual INotebook &removeTabSection(const Cursor &cursor);
```

If the specified page is a major tab page, removes the specified page and all subsequent pages up to the next major tab page. If the specified page is a minor tab page, removes the specified page and all subsequent pages up to the next page that has a tab.

resetBackgroundMajorColor

Condition: (1) Ignored on AIX

```
virtual INotebook &resetBackgroundMajorColor();
```

Resets the major tab background color by changing a color that was previously set.

resetBackgroundMinorColor

Condition: (1) Ignored on AIX

```
virtual INotebook &resetBackgroundMinorColor();
```

Resets the minor tab background color by changing a color that was previously set.

resetBackgroundPageColor

Condition: (1) Ignored on AIX

```
virtual INotebook &resetBackgroundPageColor();
```

Resets the background page color by changing a color that was previously set.

resetForegroundMajorColor

Condition: (1) Ignored on AIX

```
virtual INotebook &resetForegroundMajorColor();
```

Resets the major tab foreground color by changing a color that was previously set.

resetForegroundMinorColor

Condition: (1) Ignored on AIX

```
virtual INotebook &resetForegroundMinorColor();
```

Resets the minor tab foreground color by changing a color that was previously set.

setBackgroundColor

Condition: (3) AIX only

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

Resets the background color value of the notebook area.

setBackgroundMajorColor

```
virtual INotebook &setBackgroundMajorColor(
                     const IColor &color);
```

Resets the major tab background color by changing a color that was previously set.

setBackgroundMinorColor

```
virtual INotebook &setBackgroundMinorColor(
                     const IColor &color);
```

Resets the minor tab background color by changing a color that was previously set.

setBackgroundPageColor

```
virtual INotebook &setBackgroundPageColor(
                     const IColor &color);
```

Resets the page background color by changing a color that was previously set.

setBinding

```
virtual INotebook &setBinding(Binding binding);
```

Sets the type of binding for the notebook.

binding Use the enumeration Binding (see page 595) to specify the binding type.

setDefaultStyle

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

Sets the default style for all subsequent notebooks.

style Use the styles provided by INotebook::Styles (see page 607) to specify the default style.

setForegroundColor

Condition: (3) AIX only

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

Resets the foreground color value of the notebook area.

setForegroundMajorColor

```
virtual INotebook &setForegroundMajorColor(
                    const IColor &color);
```

Sets the major tab foreground color value of the notebook.

setForegroundMinorColor

```
virtual INotebook &setForegroundMinorColor(
                    const IColor &color);
```

Sets the minor tab foreground color value of the notebook.

setMajorTabSize

Condition: (1) Ignored on AIX

```
virtual INotebook &setMajorTabSize(
                    const ISize &sizeMajorTab);
```

Sets the size of the notebook's major tabs in pixels.

sizeMajorTab
The new size for major tabs.

setMinorTabSize

Condition: (1) Ignored on AIX

```
virtual INotebook &setMinorTabSize(
                    const ISize &sizeMinorTab);
```

Sets the size of the notebook's minor tabs in pixels.

sizeMinorTab
The new size of the minor tabs.

setOrientation

```
virtual INotebook &setOrientation(Orientation orientation);
```

Sets the location of the back page's corner and the major tabs.

orientation Use the enumeration Orientation (see page 595) to specify the orientation type.

setPageButtonSize

Condition: (1) Ignored on AIX

```
virtual INotebook &setPageButtonSize(
                    const ISize &sizePageButton);
```

Sets the size of the arrow buttons used to turn the notebook's pages, in pixels.

sizePageButton
The new size for the arrow buttons.

setStatusText

```
virtual INotebook &setStatusText(
                    const IPageHandle &referencePage,
                    const IResourceId &resourceId);

virtual INotebook &setStatusText(
                    const IPageHandle &referencePage,
                    const char *statusText);
```

Sets the status text for the specified page of the notebook.

setStatusTextAlignment

```
virtual INotebook &setStatusTextAlignment(
                    TextAlignment alignment);
```

Sets the alignment of the text in the notebook's status line.

alignment Use the enumeration TextAlignment (see page 596) to specify the alignment of the status text.

setTabBitmap

```
virtual INotebook &setTabBitmap(
                    const IPageHandle &referencePage,
                    const IBitmapHandle &bitmap);

virtual INotebook &setTabBitmap(
                    const IPageHandle &referencePage,
                    const IResourceId &resourceId);
```

Sets the tab bit map for a specified page of the notebook.

setTabShape

Condition: (1) Ignored on AIX

```
virtual INotebook &setTabShape(TabShape tabShape);
```

Sets the shape of the notebook's tabs.

tabShape Use the enumeration TabShape (see page 596) to specify the shape of the tab.

setTabText

```
virtual INotebook &setTabText(
                    const IPageHandle &referencePage,
                    const char *tabText);

virtual INotebook &setTabText(
                    const IPageHandle &referencePage,
                    const IResourceId &resourceId);
```

Sets the tab text for a specified page of the notebook.

setTabTextAlignment

```
virtual INotebook &setTabTextAlignment(
                    TextAlignment alignment);
```

Sets the alignment of the text in the notebook's tabs.

alignment Use the enumeration TextAlignment (see page 596) to specify the alignment of the tab text.

setUserData

```
virtual INotebook &setUserData(
                    const IPageHandle &referencePage,
                    unsigned long userData);
```

Sets the user data for a specified page of the notebook.

setWindow

```
virtual INotebook &setWindow(
                    const Cursor &cursor,
                    IWindow *window = 0);

virtual INotebook &setWindow(
                    const IPageHandle &referencePage,
                    IWindow *window = 0);
```

Associates the specified page with the window.

If you use a page selection handler to delay creating a window associated with a notebook page, you can use setWindow in the selection handler routine.

statusTextAlignment

```
virtual TextAlignment statusTextAlignment() const;
```

Returns the alignment of the text in the notebook's status line. The returned value is an enumerator provided by TextAlignment (see page 596).

tabShape

Condition: (1) Ignored on AIX

```
virtual TabShape tabShape() const;
```

Returns the shape of the notebook's tabs. The returned value is an enumerator provided by TabShape (see page 596).

tabTextAlignment

```
virtual TextAlignment tabTextAlignment() const;
```

Returns the alignment of the text in the notebook's tabs. The returned value is an enumerator provided by TextAlignment (see page 596).

topPage

```
virtual IPageHandle topPage() const;
```

Returns an IPageHandle (see page 634) object that references the current top page of the notebook.

totalPages

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

Returns the number of pages in the notebook.

turnToPage

```
virtual INotebook &turnToPage(const Cursor &cursor);

virtual INotebook &turnToPage(const IPageHandle &page);
```

Moves a specified page to the top of the notebook.

window

```
virtual IWindow *window(const IPageHandle &page) const;

virtual IWindow *window(const Cursor &cursor) const;
```

Returns the window associated with the specified IPageHandle.

Figure 187. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow

Figure 187. Inherited Public Members

Member function	Class
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow

Figure 187. Inherited Public Members

Member function	Class
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum recommended size for this notebook. This size is large enough to contain the largest page window that is currently in the notebook.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

unregisterCallbacks

Condition: (3) AIX only

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

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

Figure 188. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow

Figure 188. Inherited Protected Members

Member function	Class
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Binding

```
public:
enum Binding
  {
  spiral,
  solid,
  };
```

Use these enumerators to specify the binding for the notebook:

spiral
Draws a spiral binding for the notebook.

solid
Draws a solid binding for the notebook.

Orientation

```
public:
enum Orientation
  {
  backpagesBottomTabsRight,
  backpagesTopTabsRight,
  backpagesBottomTabsLeft,
  backpagesTopTabsLeft,
  backpagesRightTabsTop,
  backpagesLeftTabsTop,
  backpagesRightTabsBottom,
  backpagesLeftTabsBottom,
  };
```

Use these enumerators to specify the location of the back page's corner and the major tabs:

backpagesBottomTabsRight
Back pages on the bottom-right corner and the major tabs on the right side of the notebook.

backpagesTopTabsRight
Back pages on the top right corner and the major tabs on the right side of the notebook.

backpagesBottomTabsLeft
Back pages on the bottom-left corner and the major tabs on the left side of the notebook.

backpagesTopTabsLeft
Back pages on the top-left corner and the major tabs on the left side of the notebook.

backpagesRightTabsTop
Back pages on the top-right corner and the major tabs on the top side of the notebook.

backpagesLeftTabsTop
Back pages on the top-left corner and the major tabs on the top side of the notebook.

backpagesRightTabsBottom
Back pages on the bottom-right corner and the major tabs on the bottom side of the notebook.

backpagesLeftTabsBottom
Back pages on the bottom-left corner and the major tabs on the bottom side of the notebook.

TabShape

```
public:
enum TabShape
  {
  square,
  rounded,
  polygon,
  };
```

Use these enumerators to specify the shape of the notebook tabs:

square
Draws the tabs with square corners.

rounded
Draws the tabs with rounded corners.

polygon
Draws the tabs with polygon corners.

TextAlignment

```
public:
enum TextAlignment
  {
  left,
  right,
  center,
  };
```

Use these enumerators to specify the alignment for the text in the status line and in the tabs:

left
Left-justifies the text.

right
Right-justifies the text.

center
Centers the text.

Chapter 128. INotebook::Cursor

Derivation

IBase
  IVBase
    INotebook::Cursor

Header File: inotebk.hpp

The nested class INotebook::Cursor defines objects that you can use to traverse the pages of a notebook. In the same way that you can use a cursor to traverse the objects in a collection, you can use this cursor object to traverse a notebook, one page at a time.

INotebook::Cursor processes IPageHandle (see page 634) objects, each of which can be thought of as a pointer to a specific page in the notebook.

Constructors

```
public:
Cursor(const INotebook &notebook);
```

You can only construct objects of this class using the notebook whose pages this cursor object is going to traverse.

notebook Notebook being traversed.

Public Members

current

```
virtual IPageHandle current() const;
```

Returns an IPageHandle (see page 634) object for the notebook page to which the cursor currently points.

first

```
virtual IPageHandle first();
```

Points to the first page in the notebook and returns an IPageHandle (see page 634) object that points to the first notebook page.

© Copyright IBM Corp. 1994

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

Flags this cursor as not valid.

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

Queries whether the cursor points to a valid item.

last
```
virtual IPageHandle last();
```

Points to the last page in the notebook and returns an IPageHandle (see page 634) object that points to the last notebook page.

next
```
virtual IPageHandle next();
```

Points to the next page in the notebook and returns an IPageHandle (see page 634) object that points to the next notebook page.

previous
```
virtual IPageHandle previous();
```

Points to the previous page in the notebook and returns an IPageHandle (see page 634) object that points to the previous notebook page.

setCurrent
```
void setCurrent(const IPageHandle &current);
```

Uses a specified IPageHandle (see page 634) to set the cursor to a particular notebook page.

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

Sets the cursor to point to the first page of the notebook.

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

Sets the cursor to point to the last page of the notebook.

setToNext

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

Sets the cursor to point to the next page of the notebook.

setToPrevious

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

Sets the cursor to point to the previous page of the notebook.

Figure 189. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 129. INotebook::PageSettings

Derivation

IBase
  IVBase
    INotebook::PageSettings

Header File: inotebk.hpp

The nested class INotebook::PageSettings identifies information about a page in the notebook. You create PageSettings objects to define characteristics of pages when adding the pages to a notebook. You can use one PageSettings object to create multiple pages.

You can also create a PageSettings object to reflect the current state of a page in the notebook.

The PageSettings object only identifies the characteristics of a page at one point in time. It is not updated when the actual page changes. Also, changes to the PageSettings object are not reflected in the actual page after it is added to the notebook.

Nested Classes

INotebook::PageSettings::Attribute

Constructors

```
public:
PageSettings();

public:
PageSettings(const Attribute &attribute);

public:
PageSettings(
   const char *tabText,
   const char *statusText = 0,
   const Attribute &pageAttribute = noAttribute);

public:
PageSettings(
   const IBitmapHandle &tabBitmap,
   const char *statusText = 0,
   const Attribute &pageAttribute = noAttribute);

public:
PageSettings(const INotebook::PageSettings &);

public:
virtual PageSettings::~ PageSettings();
```


© Copyright IBM Corp. 1994

You can construct objects of this class in the following ways:

- Using the default. The User Interface Class Library defines the default attribute to be noAttribute and all the other page settings values are set to null or 0.
- Using one or more page attributes. All the other page settings are set to null or 0.
- Using the tab text, status text, and one or more attributes. Use this when there is no bit map for this tab.
- Using the tab bit map, status text, and one or more attributes. Use this when there is no text for this tab.

attribute INoteBook::PageSettings::Attribute object.

tabText Text for tab on the notebook page.

statusText Text for the status line of the notebook page.

pageAttribute
INoteBook::PageSettings::Attribute object. The default is noAttribute.

tabBitmap Bit map for the tab on the notebook page.

Public Members

isAutoSize
```
Boolean isAutoSize() const;
```

Queries whether the page's size is handled by the notebook.

isMajorTab
```
Boolean isMajorTab() const;
```

Queries whether a major tab is associated with the page.

isMinorTab
```
Boolean isMinorTab() const;
```

Queries whether a minor tab is associated with the page.

isStatusTextOn
```
Boolean isStatusTextOn() const;
```

Queries whether a status line is associated with the page.

setStatusText
```
virtual INotebook::PageSettings &setStatusText(
                                  const IResourceId &resourceId);

virtual INotebook::PageSettings &setStatusText(
                                  const char *statusText);
```

Sets the text of the status line associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

setTabBitmap

```
virtual INotebook::PageSettings &setTabBitmap(
                                    const IBitmapHandle &bitmap);

virtual INotebook::PageSettings &setTabBitmap(
                                    const IResourceId &resourceId);
```

Sets the bit map of the tab associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

setTabText

```
virtual INotebook::PageSettings &setTabText(
                                    const char *tabText);

virtual INotebook::PageSettings &setTabText(
                                    const IResourceId &resourceId);

virtual INotebook::PageSettin setTabText(
                                const char *tabText);
```

Sets the text of the tab associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

setUserData

```
virtual INotebook::PageSettings &setUserData(
                                    unsigned long userData);
```

Sets the application data in the page's reserved storage. Each page has a 4-byte reserved storage area. This area is available for information required by your application.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object

has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

statusText

```
virtual IString statusText() const;
```

Returns the text of the status line associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

tabBitmap

```
virtual IBitmapHandle tabBitmap() const;
```

Returns the bit map of the tab associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

tabText

```
virtual IString tabText() const;
```

Returns the text of the tab associated with this page.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

userData

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

Returns the application data in the page's reserved storage.

Note: Once you have inserted a page in the notebook using an INotebook::PageSettings object, using this function on the PageSettings object has no effect on the actual notebook. Instead, use the corresponding notebook

functions. However, you can use this function to update the PageSettings object for defining a new notebook page.

Figure 190. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 130. INotebook::PageSettings::Attribute

Derivation

IBase
  IBitFlag
    INotebook::PageSettings::Attribute

Header File: inotebk.hpp

The nested class INotebook::PageSettings::Attribute provides a set of valid notebook page attributes for the member functions of the class INotebook::PageSettings (see page 600).

Attributes

autoPageSize
Specifies the positioning and sizing of the notebook page. When you create a page using this attribute, the system resizes the window associated with the page so that it fills in the page area.

majorTab
Specifies that the page is associated with a major tab.

minorTab
Specifies that the page is associated with a minor tab.

noAttribute
Specifies that no attribute applies to the notebook page.

statusTextOn
Specifies that the page has a status line.

Public Members

Figure 191. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 192. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 131. INotebook::Style

Derivation

IBase
  IBitFlag
    INotebook::Style

Header File: inotebk.hpp

The nested class INotebook::Style provides a set of valid notebook styles for the static member functions of the class INotebook (see page 579). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

The combination of styles you specify for the back pages and major tabs determines how the pages in the notebook look and behave when the user pages through the notebook.

Styles

backPagesBottomLeft
Draws the back pages of the notebook behind the bottom left corner of the notebook.

backPagesBottomRight
Draws the back pages of the notebook behind the bottom right corner of the notebook. This is the default.

backPagesTopLeft
Draws the back pages of the notebook behind the top left corner of the notebook.

backPagesTopRight
Draws the back pages of the notebook behind the top right corner of the notebook.

classDefaultStyle
Specifies the original default style for this class, which is squareTabs | solidBinding | statusTextLeft | tabTextCenter | backPagesBottomRight | majorTabsRight | IWindow::visible.

majorTabsBottom
Draws the major tabs of the notebook on the bottom side of the notebook.

majorTabsLeft
Draws the major tabs of the notebook on the left side of the notebook.

© Copyright IBM Corp. 1994

majorTabsRight
Draws the major tabs of the notebook on the right side of the notebook. This is the default.

majorTabsTop
Draws the major tabs of the notebook on the top side of the notebook.

polygonTabs
Draws tabs with polygon corners.

roundedTabs
Draws tabs with rounded corners.

solidBinding
Draws a solid binding. This is the default.

spiralBinding
Draws a spiral binding.

squareTabs
Draws tabs with square corners. This is the default.

statusTextCenter
Centers the text in the status line.

statusTextLeft
Left-justifies the text in the status line. This is the default.

statusTextRight
Right-justifies the text in the status line.

tabTextCenter
Centers the text in the tabs. This is the default.

tabTextLeft
Left-justifies the text in the tabs.

tabTextRight
Right-justifies the text in the tabs.

Public Members

Figure 193. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 194. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 132. INotifier

Derivation

IBase
  IVBase
    INotifier

Header File: inotify.hpp

The class INotifier defines the protocol that objects supporting observation must supply. As this class is an abstract base class, you cannot construct objects of this class.

Public Members

disableNotification

```
virtual INotifier &disableNotification() = 0;
```

Causes the notifier to stop sending notifications to its observer objects.

enableNotification

```
virtual INotifier &enableNotification(
                    Boolean enable = true) = 0;
```

Causes the notifier to start sending notifications to observer objects. This function can be overridden by derived classes to perform any setup required for notification.

isEnabledForNotification

```
virtual Boolean isEnabledForNotification() const = 0;
```

Returns true if a notifier is sending notifications to its observers.

notifyObservers

```
virtual INotifier &notifyObservers(
                    const INotificationEvent &event) = 0;
protected:
virtual INotifier &notifyObservers(
                    const INotificationId &id) = 0;
```

Notifies all observers in a notifier's collection. Each observer receives a notification event describing the identity of the notifier, the notification id, and optional data provided by the specific notifier object.

© Copyright IBM Corp. 1994

Figure 195. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

addObserver

```
virtual INotifier &addObserver(
                      IObserver &observer,
                      const IEventData &userData) = 0;
```

Adds an observer to the notifier's collection.

notifyObservers

```
virtual INotifier &notifyObservers(
                      const INotificationId &id) = 0;
public:
virtual INotifier &notifyObservers(
                      const INotificationEvent &event) = 0;
```

Notifies all observers in a notifier's collection. Each observer receives a notification event describing the identity of the notifier, the notification id, and optional data provided by the specific notifier object.

observerList

```
virtual IObserverList &observerList() const = 0;
```

Returns the collection of observers. If the observer list does not exist, it is the implementor's responsibility to create it.

removeAllObservers

```
virtual INotifier &removeAllObservers() = 0;
```

Removes all observers from the notifier's collection.

removeObserver

```
virtual INotifier &removeObserver(
                      const IObserver &observer) = 0;
```

Removes an observer from the notifier's collection.

Chapter 133. INumericSpinButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IBaseSpinButton
            INumericSpinButton

Header File: ispinnum.hpp

Objects of the INumericSpinButton class create and manage numeric spin button control windows.

Handlers derived from the following classes can handle events for INumericSpinButton objects.

- IEditHandler (see page 247)
- IEditVerifyHandler (see page 253)
- IFocusHandler (see page 307)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IResizeHandler (see page 732)
- ISpinHandler (see page 821)
- IPaintHandler (see page 647)

Portability Considerations: 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 style.

Motif Information: The INumericSpinButton constructor creates objects of this class using several 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 INumericSpinButton object via private callbacks and a default handler. The INumericSpinButton class uses a default handler derived from the class IKeyboardHandler. Therefore, you should 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 prior to the default handler.

© Copyright IBM Corp. 1994

Nested Classes

INumericSpinButton::Style

Constructors

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

Condition: (2) OS/2 Only

```
public:
INumericSpinButton(unsigned long id, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
INumericSpinButton(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Creating the specified numeric spin button control and an object for it

  windowId A unique ID for the numeric spin button control.

  parent The parent window.

  owner The owner window.

  initial (Optional) A rectangle defining the size and placement of the numeric spin button window.

  style (Optional) The style of the control.

- Creating the object for the specified numeric spin button control.

  id A unique ID for the numeric spin button control.

  parent The parent dialog window.

- Creating the object for the specified numeric spin button control

  handle The window handle of an existing numeric spin button control.

Public Members

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 619) unless you have changed the style using setDefaultStyle (see page 614).

isSpinFieldValid

```
virtual Boolean isSpinFieldValid(
                 Boolean caseSensitive = false) const;
```

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.

range

```
virtual IRange range() const;
```

Returns the number range of the spin button.

setDefaultStyle

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

Sets the default style for all subsequent numeric spin buttons.

style Use the styles provided by INumericSpinButton::Styles (see page 619) to specify the default style.

setLimit

Condition: (3) AIX only

```
virtual INumericSpinButton &setLimit(unsigned long aNumber);
```

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.

setRange

```
virtual INumericSpinButton &setRange(
                            const IRange &aNewRange,
                            Boolean override = false);
```

Sets the lower and upper limit of the spin button range.

aNewRange
The new lower and upper limit.

override The default is false. 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.

setValue

```
virtual INumericSpinButton &setValue(long aValue);
```

Sets the displayed value of the spin field, regardless of the validity of the number.

spinDown

Condition: (3) AIX only

```
virtual INumericSpinButton &spinDown(
                              unsigned long spinBy = 1);
```

Spins the button down the specified number of times.

spinTo

```
virtual INumericSpinButton &spinTo(
                              long aValue,
                              Boolean spinToClosest = false);
```

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.

spinUp

Condition: (3) AIX only

```
virtual INumericSpinButton &spinUp(
                              unsigned long spinBy = 1);
```

Spins the button up the specified number of times.

value

```
virtual long value() const;
```

Returns the current value displayed in the spin field.

Figure 196. Inherited Public Members

Member function	Class
activeColor	IWindow
addBorder	IBaseSpinButton
alignment	IBaseSpinButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disableDataUpdate	IBaseSpinButton
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableFastSpin	IBaseSpinButton
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enable	IBaseSpinButton
enableDataUpdate	IBaseSpinButton
enableFastSpin	IBaseSpinButton
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IBaseSpinButton
handle	IWindow
handleException	IWindow
hasBorder	IBaseSpinButton
hasFocus	IWindow

Figure 196. Inherited Public Members

Member function	Class
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFastSpinEnabled	IBaseSpinButton
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isMaster	IBaseSpinButton
isServant	IBaseSpinButton
isShowing	IWindow
isSpinFieldValid	IBaseSpinButton
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
isWriteable	IBaseSpinButton
itemProvider	IWindow
layoutAdjustment	IWindow
limit	IBaseSpinButton
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveSizeTo	IBaseSpinButton
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow

Figure 196. Inherited Public Members

Member function	Class
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeBorder	IBaseSpinButton
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IBaseSpinButton
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow

Figure 196. Inherited Public Members

Member function	Class
setLimit	IBaseSpinButton
setMaster	IBaseSpinButton
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
spinDown	IBaseSpinButton
spinUp	IBaseSpinButton
topHandle	IWindow
topHandle	IBaseSpinButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

registerCallbacks

Condition: (3) AIX only

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

Called during construction of the INumericSpinButton window. This function registers the Motif callbacks for the underlying widgets.

unregisterCallbacks

Condition: (3) AIX only

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

Called during destruction of the INumericSpinButton window. This function unregisters the Motif callbacks for the underlying widgets.

Figure 197. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
calcMinimumSize	IBaseSpinButton
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
initialize	IBaseSpinButton
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IBaseSpinButton
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier

Figure 197. Inherited Protected Members

Member function	Class
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IBaseSpinButton
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 134. INumericSpinButton::Style

Derivation

IBase
  IBitFlag
    INumericSpinButton::Style

Header File: ispinnum.hpp

The nested class INumericSpinButton::Style provides a set of valid spin button styles for objects of the class INumericSpinButton (see page 612). You can use these styles in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IBaseSpinButton::Style (see page 40)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IBaseSpinButton::classDefaultStyle.

padWithZeros
Pads the displayed number with zeros at the front.

Public Members

Figure 198. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 199. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 135. IObjectWindow

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IObjectWindow

Header File: iobjwin.hpp

Objects of the IObjectWindow class create 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.

Motif Information: 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.

Presentation Manager Information: These windows have minimal system overhead because they are created with a parent of HWND_OBJECT.

Constructors

```
public:
IObjectWindow();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters. The default constructor creates an IObjectWindow that starts handling events immediately.

Public Members

Figure 200. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*

Figure 200. Inherited Public Members

Member function	Class
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow

© Copyright IBM Corp. 1994

Figure 200. Inherited Public Members

Member function	Class
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
enable	IWindow
enableNotification	INotifier
enableNotification	IWindow
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow

Figure 200. Inherited Public Members

Member function	Class
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow

Figure 200. Inherited Public Members

Member function	Class
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 201. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow

Figure 201. Inherited Protected Members

Member function	Class
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 136. IOutlineBox

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          IOutlineBox
```

Header File: ioutlbox.hpp

Objects of the IOutlineBox class create and manage an unfilled rectangle control.

Nested Classes

IOutlineBox::Style

Constructors

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

Condition: (2) OS/2 Only

```
public:
IOutlineBox(unsigned long id, IWindow *parentDialog);
```

Condition: (2) OS/2 Only

```
public:
IOutlineBox(const IWindowHandle &handle);
```

You can create an outline box control object in the following ways:

- Using the parent window, owner window, optional size and location, and optional style parameters.
- Using the parent window dialog.
- Using the handle of an existing outline box control.

id An outline box control ID.

parent The parent window.

owner The owner window.

© Copyright IBM Corp. 1994

initial (Optional) A rectangle for the outline box control. It specifies the initial position and size of the outline box you are constructing.

style (Optional) The initial style for the outline box control. The default is classDefaultStyle

parentDialog
The parent dialog window.

handle An existing outline box window handle.

Public Members

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

Returns the default style. The default style is classDefaultStyle (see page 630) unless you have changed the style using setDefaultStyle (see page 626).

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

Returns the foreground color value of the outline box, or the default if no color for the outline box has been set.

moveSizeTo
Condition: (3) AIX only

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

Changes the position and size of the outline box.

outlineType
```
OutlineType outlineType() const;
```

Returns the current type of outline for this outline box object. The returned value is one of the enumerators provided by OutlineType (see page 629).

position
Condition: (3) AIX only

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

Returns the position of the outline box.

setDefaultStyle

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

Sets the default style for all subsequent outline boxes.

style Use the styles provided by IOutlineBox::Styles (see page 630) to specify the default style.

setForegroundColor

Condition: (3) AIX only

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

Sets the foreground color of the outline box.

setOutlineType

```
IOutlineBox &setOutlineType(const OutlineType &type);
```

Sets the type of outline for this outline box object.

type Enumerated type of outline. Use the enumerators provided by OutlineType (see page 629) to specify the type of outline to set.

size

Condition: (3) AIX only

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

Returns the size of the outline box.

Figure 202. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow

Figure 202. Inherited Public Members

Member function	Class
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow

Figure 202. Inherited Public Members

Member function	Class
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow

Figure 202. Inherited Public Members

Member function	Class
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow

Figure 202. Inherited Public Members

Member function	Class
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 203. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow

Figure 203. Inherited Protected Members

Member function	Class
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

OutlineType

```
public:
enum OutlineType
  {
  foreground,
  background,
  halftone,
  };
```

Use these enumerators to specift outline types:

foreground
Specifies the currently set foreground color.

background
Specifies the system's background color.

halftone
Specifies halftone shading in the foreground color.

Chapter 137. IOutlineBox::Style

Derivation

IBase
  IBitFlag
    IOutlineBox::Style

Header File: ioutlbox.hpp

The nested class IOutlineBox::Style provides a set of valid outline box styles for the static member functions of the class IOutlineBox (see page 624). You can use these styles with the styles defined by the following nested class:

IWindow::Style (see page 957)

Styles

backgroundFrame
Causes the outline box to have a frame equal to the background color.

classDefaultStyle
Specifies the default style for this class, which is foregroundFrame | IWindow::visible.

foregroundFrame
Causes the outline box to have a frame equal to the foreground color. This is the default style.

halftoneFrame
Causes the outline box to have the frame shaded in a halftone of the foreground color.

Motif Information: The AIX release ignores the halftoneFrame.

Public Members

Figure 204. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*

Figure 204. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 205. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 138. IPageEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IPageEvent

Header File: ipageevt.hpp

Objects of the IPageEvent class represent a notification for a page-related notebook event. A notebook page handler creates an IPageEvent object when a notebook page is resized. The IPageEvent class is also the base class for other page-related events.

Note: Page events are first dispatched to the notebook, then to the owner window of the notebook.

Constructors

```
public:
IPageEvent(const IEvent &event);

public:
IPageEvent(const IControlEvent &controlEvent);
```

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (see page 636) creates objects of this class from an object of the class IEvent (see page 275) or IControlEvent (see page 227).

Public Members

notebook

```
virtual INotebook *notebook() const;
```

Returns the notebook object the page belongs to.

pageHandle

```
virtual IPageHandle pageHandle() const;
```

Returns the notebook page handle that is the subject of the notification event.

Figure 206. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*

Figure 206. Inherited Public Members

Member function	Class
asString	IBase*
asString	IVBase*

© Copyright IBM Corp. 1994

Figure 206. Inherited Public Members

Member function	Class
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 139. IPageHandle

Derivation

IBase
  IHandle
    IPageHandle

Header File: inotebk.hpp

Objects of the IPageHandle class represent a page in a notebook. The User Interface Class Library only enables notebooks to construct IPageHandle objects. A notebook returns an IPageHandle when:

- A page is added to the notebook. The IPageHandle represents a new page.
- A page is queried, such as from a call to INotebook::firstPage (see page 583).

You can use an IPageHandle object to identify a page, just as you can with an INotebook::Cursor (see page 597) object.

Constructors

```
public:
IPageHandle(Value pageId = 0);
```

You can construct objects of this class from a page identifier (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the page handle value as a type.

Figure 207. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 140. IPageHandler

Derivation

IBase
  IVBase
    IHandler
      IPageHandler

Header File: ipagehdr.hpp

Objects of the IPageHandler class handle the processing of page change events for the INotebook (see page 579) class. These events result from a user interacting with a notebook page.

You can create a handler derived from IPageHandler and attach it to a notebook or to the notebook's owner window. You do this by calling IHandler::handleEventsFor (see page 403) to pass the notebook or owner window to the page handler.

IPageHandler dispatches the following events:

- Requesting help for a page: IPageHelpEvent (see page 638)
- Removing a page: IPageRemoveEvent (see page 640)
- Resizing a page: IPageEvent (see page 632)
- Selecting a page: IPageSelectEvent (see page 642)

When the page handler receives one of these events, it creates a corresponding event object and routes that object to the appropriate IPageHandler virtual function. You can override these virtual functions to supply customized processing of a page event.

The return value from the virtual functions specifies whether the page event is passed on for additional processing, as follows:

true The page event requires no additional processing. Do not pass it to another handler.

false Pass the page event to the next handler for additional processing, as follows:

- If there is another handler for the notebook, pass the page event to the next handler.
- If this is the last handler for the notebook, call IWindow::dispatch (see page 947) to dispatch the page event to the notebook's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (see page 946) to process the page event.

© Copyright IBM Corp. 1994

Constructors

```
public:
IPageHandler();
```

You can only construct objects of this class using the default constructor, which does not accept any parameters.

Public Members

Figure 208. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a page event is received, the appropriate virtual function is called.

drawTab

Condition: (2) OS/2 Only

```
virtual Boolean drawTab(INotebookDrawItemEvent &event);
```

This class is used by derived classes to draw the page tab.

help

Condition: (1) Ignored on AIX

```
virtual Boolean help(IPageHelpEvent &event);
```

This class is used by derived classes to process a notification that help has been requested for a notebook tab.

Motif Information: You can set up help for individual notebook pages by setting up the help table and IHelpWindow object for each page in exactly the same way as any other window object.

remove

Condition: (1) Ignored on AIX

```
virtual Boolean remove(IPageRemoveEvent &event);
```

Implemented by derived classes to process a page deletion notification.

resize

Condition: (1) Ignored on AIX

```
virtual Boolean resize(IPageEvent &event);
```

Implemented by derived classes to process a page size change notification.

select

```
virtual Boolean select(IPageSelectEvent &event);
```

Implemented by derived classes to process the notification that a new page is at the top of the notebook.

Figure 209. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 141. IPageHelpEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IPageEvent
          IPageHelpEvent

Header File: ipageevt.hpp

Objects of the IPageHelpEvent class represent a notification that help has been requested for a notebook page. A notebook page handler creates and uses an IPageHelpEvent object.

Note: These events are first dispatched to the notebook, then to the owner window of the notebook.

Motif Information: The AIX release of the User Interface Class Library does not support this class.

Constructors

Condition: (1) Ignored on AIX

```
public:
IPageHelpEvent(const IEvent &event);
```

Condition: (1) Ignored on AIX

```
public:
IPageHelpEvent(const IControlEvent &controlEvent);
```

Condition: (1) Ignored on AIX

```
public:
IPageHelpEvent(const IPageEvent &pageEvent);
```

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (see page 636) creates objects of this class from an object of the class IEvent (see page 275), IControlEvent (see page 227), or IPageEvent (see page 632).

Public Members

© Copyright IBM Corp. 1994

helpWindow

Condition: (1) Ignored on AIX

```
IHelpWindow *helpWindow() const;
```

Returns the help window you can use to display help for the notebook page.

notebook

Condition: (1) Ignored on AIX

```
virtual INotebook *notebook() const;
```

Returns the notebook object the page belongs to.

pageHandle

Condition: (1) Ignored on AIX

```
virtual IPageHandle pageHandle() const;
```

Returns the handle of the notebook page for which help has been requested.

Figure 210. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
notebook	IPageEvent
operator <<	IBase*
operator <<	IVBase*
pageHandle	IPageEvent
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent

Figure 210. Inherited Public Members

Member function	Class
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 142. IPageRemoveEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IPageEvent
          IPageRemoveEvent

Header File: ipageevt.hpp

Objects of the IPageRemoveEvent class represent a notification of the deletion of a notebook page. A notebook page handler creates and uses an IPageRemoveEvent object.

These events provide functions to return the application page window and bit map. The application must delete the window and bit map, if any.

Note: These events are first dispatched to the notebook, then to the owner window of the notebook.

Motif Information: The AIX release of the User Interface Class Library does not support this class.

Constructors

Condition: (1) Ignored on AIX

```
public:
IPageRemoveEvent(const IEvent &event);
```

Condition: (1) Ignored on AIX

```
public:
IPageRemoveEvent(const IControlEvent &controlEvent);
```

Condition: (1) Ignored on AIX

```
public:
IPageRemoveEvent(const IPageEvent &pageEvent);
```

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (see page 636) creates objects of this class from an object of the class IEvent (see page 275), IControlEvent (see page 227), or IPageEvent (see page 632).

© Copyright IBM Corp. 1994

Public Members

notebook

Condition: (1) Ignored on AIX

```
virtual INotebook *notebook() const;
```

Returns the notebook object that the page belongs to.

pageWindow

Condition: (1) Ignored on AIX

```
IWindow *pageWindow() const;
```

Returns the window or control associated with the page.

tabBitmap

Condition: (1) Ignored on AIX

```
IBitmapHandle tabBitmap() const;
```

Returns the handle of the bit map used by the tab associated with the page.

Figure 211. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
notebook	IPageEvent
operator <<	IBase*
operator <<	IVBase*
pageHandle	IPageEvent
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent

Figure 211. Inherited Public Members

Member function	Class
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 143. IPageSelectEvent

Derivation

IBase
  IVBase
    IEvent
      IControlEvent
        IPageEvent
          IPageSelectEvent

Header File: ipageevt.hpp

Objects of the IPageSelectEvent class represent a notification that a page has been selected in a notebook. A notebook page handler creates and uses an IPageSelectEvent object.

Note: These events are first dispatched to the notebook, then to the owner window of the notebook.

Constructors

```
public:
IPageSelectEvent(const IEvent &event);

public:
IPageSelectEvent(const IControlEvent &controlEvent);

public:
IPageSelectEvent(const IPageEvent &pageEvent);
```

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (see page 636) creates objects of this class from an object of the class IEvent (see page 275), IControlEvent (see page 227), or IPageEvent (see page 632).

Public Members

notebook

```
virtual INotebook *notebook() const;
```

Returns the notebook object the page belongs to.

pageHandle

```
virtual IPageHandle pageHandle() const;
```

Returns the handle of the selected notebook page.

© Copyright IBM Corp. 1994

previousSelectedPageHandle

```
IPageHandle previousSelectedPageHandle() const;
```

Returns the handle of the previously selected notebook page.

Figure 212. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlId	IControlEvent
controlWindow	IEvent
controlWindow	IControlEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
notebook	IPageEvent
operator <<	IBase*
operator <<	IVBase*
pageHandle	IPageEvent
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 144. IPaintEvent

Derivation

IBase
  IVBase
    IEvent
      IPaintEvent

Header File: ipainevt.hpp

Objects of the IPaintEvent class represent a OS/2 Presentation Manager paint event, which requires a window or control to update its appearance on the screen. An object of IPaintHandler (see page 647) creates and passes this event to its virtual function IPaintHandler::paintWindow (see page 648) for processing. You must make all window updates using the presentation space handle supplied by the IPaintEvent object.

Motif Information: 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 which 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.

Constructors

```
public:
IPaintEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IPaintHandler::dispatchHandlerEvent (see page 648) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

clearBackground

© Copyright IBM Corp. 1994

```
IPaintEvent &clearBackground(
               const IRectangle &fillRectangle,
               const IColor &background = IGUIColor ( IGUIColor::desktopBgnd ));

IPaintEvent &clearBackground(
               const IColor &background = IGUIColor ( IGUIColor::desktopBgnd ));
```

Clears the specified rectangle, filling it with the specified color. If you do not specify a rectangle, this function uses the screen rectangle for the invalidated part of the window.

Motif Information: If you do not specify a rectangle, 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 (see page 935). If you override this function, you can get the graphic context using the same function. You can then:

- Change other attributes of the graphic context before calling the inherited function
- Paint the screen without changing the graphic context

If you use presSpace, be sure to call IWindow::releasePresSpace (see page 936) when you are done.

As for all coordinates and rectangles in the User Interface Class Library, this function converts the rectangle which 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

```
IPaintEvent &drawText(
               const char *text,
               const IPoint &atPoint,
               const IColor &textColor = IGUIColor ( IGUIColor::windowStaticText ));
```

Draws the specified text, beginning at the specified point, using the specified color for the text.

Motif Information: In X/Motif, the library uses the graphic context (GC) obtained via IWindow::presSpace (see page 935). If you override this function, you can get the graphic context using the same function. You can then:

- Change other attributes of the graphic context before calling the inherited function
- Paint the screen without changing the graphic context

If you use presSpace, be sure to call IWindow::releasePresSpace (see page 936) when you are done.

This member function uses X library drawing functions such as XSetForeground and XDrawString.

presSpaceHandle

```
virtual IPresSpaceHandle presSpaceHandle();
```

Returns the handle of the presentation space to use for any drawing.

rect

```
IRectangle rect() const;
```

Returns the screen rectangle for the invalidated part of the window.

setGraphicContext

```
IPaintEvent &setGraphicContext(
                const IGraphicContext &context,
                Boolean setClipRegion = true);

IPaintEvent &setGraphicContext(
                const IPresSpaceHandle &handle,
                Boolean setClipRegion = true);
```

Sets the graphic context used for drawing.

Figure 213. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 145. IPaintHandler

Derivation

```
IBase
  IVBase
    IHandler
      IPaintHandler
```

Header File: ipainhdr.hpp

Objects of the IPaintHandler class process paint events for any window or control that require 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 (see page 403) to pass the window or control to the paint handler.

When the paint handler receives a paint event, it creates an IPaintEvent (see page 644) object and routes that object to the IPaintHandler::paintWindow (see page 648) 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 (see page 946) to process the paint event.

Motif Information: Objects of the IPaintEvent class represent a Motif expose event.

Constructors

```
protected:
IPaintHandler();
```

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.

© Copyright IBM Corp. 1994

Public Members

Figure 214. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a paint event is received, the appropriate virtual function is called.

paintWindow

```
virtual Boolean paintWindow(IPaintEvent &event) = 0;
```

Implemented by derived classes to process paint events.

Figure 215. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 146. IPair

Derivation

IBase
  IPair

Header File: ipoint.hpp

Objects of the IPair class are generic ordered pairs of coordinates. The class serves as the base for the following specific ordered pair classes:

- IPoint (see page 655)
- ISize (see page 790)
- IRange (see page 715)

This class provides basic utilities to get and set the two coordinate values. In addition, it provides a full set of comparison and mathematical operators to manipulate ordered pairs.

Constructors

```
public:
IPair();

public:
IPair(Coord init);

public:
IPair(Coord coord1, Coord coord2);
```

You can construct objects of this class from the following:

- The default constructor. This constructs an ordered pair (0,0) object.
- A single coordinate. This constructs an ordered pair with both coordinates set to the same specified value.
- Two coordinates. This constructs an ordered pair with the coordinates set to the specified values.

Public Members

asDebugInfo

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

Converts the ordered pair to an IString containing a diagnostic representation of the object.

© Copyright IBM Corp. 1994

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

Converts the ordered pair (a, b) to an IString(“(a, b)”).

coord1
```
Coord coord1() const;
```

Obtains the value of the first coordinate.

coord2
```
Coord coord2() const;
```

Obtains the value of the second coordinate.

distanceFrom
```
double distanceFrom(const IPair &aPair) const;
```

Returns the distance from some other ordered pair.

dotProduct
```
long dotProduct(const IPair &aPair) const;
```

Returns the dot product with another ordered pair.

maximum
```
IPair maximum(const IPair &aPair) const;
```

Returns an ordered pair whose coordinates are the maximum of the corresponding coordinates of the IPair and the specified IPair.

minimum
```
IPair minimum(const IPair &aPair) const;
```

Returns an ordered pair whose coordinates are the minimum of the corresponding coordinates of the IPair and the specified IPair.

operator !=
```
Boolean operator !=(const IPair &aPair) const;
```

True if either coordinate differs.

operator %
```
friend IPair operator %(const IPair &aPair1, long divisor);

IPair operator %(const IPair &pair1, const IPair &pair2);

IPair operator %(const IPair &aPair1, long divisor);

friend IPair operator %(
               const IPair &pair1,
```

```
              const IPair &pair2);
```

Returns an ordered pair whose coordinates are the remainder of the corresponding coordinates of the IPair (*pair1* or *aPair1*) and the second parameter.

If the second parameter is:

pair2
: The library performs the remainder function between the corresponding coordinates, coord1 of *pair1* with coord1 of *pair2* and coord2 with coord2.

divisor
: The library performs the remainder function between each coordinate of *aPair1* and the *divisor*.

operator %=

```
IPair &operator %=(long divisor);

IPair &operator %=(const IPair &aPair);
```

Replaces the coordinates with the remainder when divided by those of the following specified parameter:

aPair
: The library performs the remainder function between the corresponding coordinates, coord1 with coord1 of *aPair* and coord2 with coord2.

divisor
: The library performs the remainder function between each coordinate and the *divisor*.

operator *

```
friend IPair operator *(
               const IPair &pair1,
               const IPair &pair2);

IPair operator *(const IPair &pair1, const IPair &pair2);

IPair operator *(const IPair &pair1, double multiplier);

friend IPair operator *(
               const IPair &pair1,
               double multiplier);
```

Returns an ordered pair whose coordinates are the product of *pair1* and the second specified parameter:

pair2
: The library performs the products function between the corresponding coordinates, coord1 of *pair1* with coord1 of *pair2* and coord2 with coord2.

multiplier
: The library performs the products function between each coordinate of *pair1* and the *multiplier*.

operator *=

```
IPair &operator *=(double multiplier);

IPair &operator *=(const IPair &aPair);
```

Multiplies the coordinates by those of the specified parameter:

aPair
: The library performs the product function between the corresponding coordinates, coord1 with coord1 of *aPair* and coord2 with coord2.

multiplier
: The library perform the product function between each coordinate and the *multiplier*.

operator +

```
friend IPair operator +(
              const IPair &pair1,
              const IPair &pair2);
```

Returns an ordered pair whose coordinates are the sums of the corresponding coordinates of *pair1* and *pair2*.

operator +=

```
IPair &operator +=(const IPair &aPair);
```

Adds the coordinates of the specified *aPair* to the coordinates of an ordered pair.

operator -

```
friend IPair operator -(
              const IPair &pair1,
              const IPair &pair2);

IPair operator -() const;

IPair operator -(const IPair &pair1, const IPair &pair2);
```

Returns an ordered pair whose coordinates are the difference between the corresponding coordinates of *pair1* and *pair2*.

When you use the unary format, it returns an ordered pair with negated coordinates.

operator -=

```
IPair &operator -=(const IPair &aPair);
```

Subtracts the coordinates specified in *aPair* from the IPair coordinates.

operator /

```
friend IPair operator /(const IPair &pair1, double divisor);

IPair operator /(const IPair &pair1, const IPair &pair2);

IPair operator /(const IPair &pair1, double divisor);

friend IPair operator /(
```

```
                const IPair &pair1,
                const IPair &pair2);
```

Returns an ordered pair whose coordinates are the quotient of the corresponding coordinates of *pair1* and the second specified parameter:

pair2
: The library performs the quotient function between the corresponding coordinates, coord1 of *pair1* with coord1 of *pair2* and coord2 with coord2.

divisor
: The library performs the quotient function between each coordinate of *pair1* and the *divisor*.

operator /=

```
IPair &operator /=(double divisor);

IPair &operator /=(const IPair &aPair);
```

Divides the coordinates by those of the second specified parameter:

aPair
: The library performs the quotient function between the corresponding coordinates, coord1 with coord1 of *aPair* and coord2 with coord2.

divisor
: The library performs the product function between each coordinate and the *divisor*.

operator <

```
Boolean operator <(const IPair &aPair) const;
```

True if both coordinates are less than those of the specified *aPair*.

operator <<

```
friend ostream &operator <<(
                    ostream &aStream,
                    const IPair &aRectangle);
```

The library provides this operator to let you write ordered pairs to ostreams in the conventional manner.

operator <=

```
Boolean operator <=(const IPair &aPair) const;
```

True if both coordinates are less than or equal.

operator ==

```
Boolean operator ==(const IPair &aPair) const;
```

True if both coordinates match those of the specified *aPair*.

operator >
```
Boolean operator >(const IPair &aPair) const;
```

True if both coordinates are greater than those of the specified *aPair*.

operator >=
```
Boolean operator >=(const IPair &aPair) const;
```

True if both coordinates are greater than or equal.

scaleBy
```
IPair &scaleBy(double xFactor, double yFactor);
```

Scales the X-coordinate by *xFactor*, the Y-coordinate by *yFactor*.

scaledBy
```
IPair scaledBy(double xFactor, double yFactor) const;
```

Same as IPair::scaleBy (see page 654), but returns a new IPair, leaving the original unmodified.

setCoord1
```
IPair &setCoord1(Coord coord1);
```

Sets the value of the first coordinate.

setCoord2
```
IPair &setCoord2(Coord coord2);
```

Sets the value of the second coordinate.

transpose
```
friend IPair transpose(const IPair &aPair);

IPair &transpose();
```

Swaps the coordinates of the ordered pair. The friend version of this function returns a new pair with transposed coordinates.

Figure 216. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 147. IPoint

Derivation

IBase
  IPair
    IPoint

Header File: ipoint.hpp

Objects of the IPoint class represent points in two-dimensional space. In addition to all the functions inherited from its base class, IPair (see page 649), the IPoint class provides additional functions.

Presentation Manager Information: You can also construct objects of this class from a Presentation Manager Toolkit POINTL structure.

Constructors

```
public:
IPoint();

public:
IPoint(const IPair &pair);

public:
IPoint(Coord x, Coord y);
```

Condition: (2) OS/2 Only

```
public:
IPoint(const struct _POINTL &ptl);
```

You can construct objects of this class in the following ways:

- From the default constructor. The resulting point has X- and Y-coordinates equal to 0.
- From any other ordered pair.
- From arbitrary X- and Y-coordinates.

Public Members

asPOINTL

Condition: (2) OS/2 Only

```
struct _POINTL asPOINTL() const;
```

Renders the point as a Presentation Manager Toolkit POINTL structure.

© Copyright IBM Corp. 1994

setX
```
IPoint &setX(Coord X);
```

Sets the point's X-coordinate.

setY
```
IPoint &setY(Coord Y);
```

Sets the point's Y-coordinate.

x
```
Coord x() const;
```

Returns the point's X-coordinate.

y
```
Coord y() const;
```

Returns the point's Y-coordinate.

Figure 217. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IPair
asString	IBase*
asString	IPair
coord1	IPair
coord2	IPair
distanceFrom	IPair
dotProduct	IPair
maximum	IPair
minimum	IPair
operator !=	IPair
operator %	IPair
operator %=	IPair
operator *	IPair
operator *=	IPair
operator +	IPair
operator +=	IPair
operator -	IPair
operator -=	IPair
operator /	IPair
operator /=	IPair
operator <	IPair
operator <<	IBase*
operator <<	IPair

Figure 217. Inherited Public Members

Member function	Class
operator <=	IPair
operator ==	IPair
operator >	IPair
operator >=	IPair
scaleBy	IPair
scaledBy	IPair
setCoord1	IPair
setCoord2	IPair
transpose	IPair

Note: * Application Support Class Library

Chapter 148. IPointerHandle

Derivation

IBase
  IHandle
    IPointerHandle

Header File: ihandle.hpp

Objects of the IPointerHandle class access and manage 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 bit map (a pointer is a type of bit map) in multiple places, and the library maintains the lifetime of this bit map until all users are finished with it.

Motif Information: IPointerHandle is an alias for the XLib Pixmap type.

Presentation Manager Information: IPointerHandle is an alias for the OS/2 Programmer's Toolkit type HPOINTER.

Constructors

```
public:
IPointerHandle(Value hpointer = 0);

public:
IPointerHandle(const IPointerHandle &aHandle);
```

You can construct objects of this class from a pointer handle (a value of type IHandle::Value), which defaults to 0. The class also provides a copy constructor and destructor in order to track references to the associated pointer resource.

Public Members

operator =

```
IPointerHandle &operator =(const IPointerHandle &aHandle);
```

Assigns the value of one pointer handle to another.

operator Value

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

Returns the IPointerHandle value.

© Copyright IBM Corp. 1994

Figure 218. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Chapter 149. IPopUpMenu

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IMenu
          IPopUpMenu

Header File: ipopmenu.hpp

Objects of the IPopUpMenu class create a pop-up menu. The User Interface Class Library supports the creation of pop-up menus on demand using IMenuHandler::makePopUpMenu (see page 510).

ICommandEvents do not specify whether they are for a menu bar or pop-up menu. If you need to tell the difference, such as when applying different actions to a menu item object when the user selects it from the menu bar as opposed to the pop-up menu, you must use different IDs for the menu items to distinguish whether they are on the menu bar or the pop-up menu.

Motif Information: The User Interface Class Library adds an X/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

Constructors

```
public:
IPopUpMenu(
   IWindow *owner,
   unsigned long menuWindowId,
   const Style &style = defaultStyle ( ));

public:
IPopUpMenu(const IResourceId &menuResId, IWindow *owner);
```

You can construct objects of this class in the following ways:

- By creating an empty pop-up menu using a window ID, frame window owner, and styles.
- By creating a pop-up menu with a menu resource ID and frame window owner.

Create the pop-up menu with the container as parent and owner and add the menu handler to the container. You must specify the container as owner of the pop-up menu for container objects.

© Copyright IBM Corp. 1994

Public Members

show

```
virtual IPopUpMenu &show(const IPoint &atLocation);
```

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 (see page 506). If you call IWindow::setAutoDeleteObject(true) (see page 938), the IPopUpMenu object is deleted when the pop-up menu ends.

topHandle

Condition: (3) AIX only

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

Returns the menu shell (parent widget) of the pop-up menu. The IWindowHandle returned is used for positioning and managing the pop-up menu.

Figure 219. Inherited Public Members

Member function	Class
activeColor	IWindow
add	IMenu
addAsNext	IMenu
addAt	IMenu
addBitmap	IMenu
addItem	IMenu
addNextAt	IMenu
addSeparator	IMenu
addSubmenu	IMenu
addText	IMenu
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IMenu
borderColor	IWindow
characterSize	IWindow
checkItem	IMenu

Figure 219. Inherited Public Members

Member function	Class
childAt	IWindow
cursor	IMenu
defaultPushButton	IWindow
deleteAt	IMenu
deleteItem	IMenu
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IMenu
disabledForegroundColor	IWindow
disabledForegroundColor	IMenu
disableItem	IMenu
disableNotification	INotifier
disableNotification	IWindow
disableUpdate	IWindow
elementAt	IMenu
enable	IWindow
enableItem	IMenu
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IMenu
enableUpdate	IWindow

Figure 219. Inherited Public Members

Member function	Class
font	IWindow
foregroundColor	IWindow
foregroundColor	IMenu
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IMenu
hiliteForegroundColor	IWindow
hiliteForegroundColor	IMenu
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isItemChecked	IMenu
isItemDisabled	IMenu
isItemEnabled	IMenu
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVerticalFlip	IMenu
isVisible	IWindow
itemProvider	IWindow
itemRect	IMenu
layoutAdjustment	IWindow
matchForMnemonic	IWindow
menuItem	IMenu
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow

Figure 219. Inherited Public Members

Member function	Class
notifyObservers	INotifier
notifyObservers	IWindow
numberOfItems	IMenu
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeConditionalCascade	IMenu
removeSubmenu	IMenu
removeSubmenuAt	IMenu
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBackgroundColor	IMenu
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledBackgroundColor	IMenu
resetDisabledForegroundColor	IWindow
resetDisabledForegroundColor	IMenu
resetForegroundColor	IWindow
resetForegroundColor	IMenu
resetHiliteBackgroundColor	IWindow
resetHiliteBackgroundColor	IMenu
resetHiliteForegroundColor	IWindow
resetHiliteForegroundColor	IMenu
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
selectItem	IMenu
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow

Figure 219. Inherited Public Members

Member function	Class
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBackgroundColor	IMenu
setBitmap	IMenu
setBorderColor	IWindow
setConditionalCascade	IMenu
setDisabledBackgroundColor	IWindow
setDisabledBackgroundColor	IMenu
setDisabledForegroundColor	IWindow
setDisabledForegroundColor	IMenu
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setForegroundColor	IMenu
setHiliteBackgroundColor	IWindow
setHiliteBackgroundColor	IMenu
setHiliteForegroundColor	IWindow
setHiliteForegroundColor	IMenu
setId	IWindow
setInactiveColor	IWindow
setItem	IMenu
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setSubmenu	IMenu
setText	IMenu
setVerticalFlip	IMenu
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
uncheckItem	IMenu
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

registerCallbacks

Condition: (3) AIX only

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

Adds X/Motif callbacks for the XmPopUpMenu row column widget.

unregisterCallbacks

Condition: (3) AIX only

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

Removes X/Motif callbacks from the XmPopUpMenu row column widget.

Figure 220. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow

Figure 220. Inherited Protected Members

Member function	Class
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 150. IPresSpaceHandle

Derivation

IBase
  IHandle
    IPresSpaceHandle

Header File: ihandle.hpp

Objects of the IPresSpaceHandle class access a presentation space.

Motif Information: IPresSpaceHandle is an alias for the GC (Graphics Context) type.

Presentation Manager Information: IPresSpaceHandle is an alias for the OS/2 Programmer's Toolkit type HPS.

Constructors

```
public:
IPresSpaceHandle(Value hps = 0);
```

Condition: (3) AIX only

```
public:
IPresSpaceHandle(_XGC *hps);
```

Condition: (3) AIX only

```
public:
IPresSpaceHandle(int hps);
```

You can construct objects of this class from a presentation space handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

Condition: (2) OS/2 Only

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

Returns the IPresSpaceHandle value.

operator _XGC *

Condition: (3) AIX only

```
operator _XGC *() const;
```


© Copyright IBM Corp. 1994

Returns the handle value as an X Graphics Context.

Figure 221. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Chapter 151. IPrivateResource

Derivation

```
IBase
  IVBase
    IResource
      IPrivateResource
```

Header File: ireslock.hpp

Use the IPrivateResource class to define a resource that is used within a single process.

IResource and its inherited classes represent resources in the users 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. They create an IResource inherited class to represent the window list and then using IResourceLock (preferred) or a function of IResource itself to lock and unlock the resource so as to serialize the access. You can use IPrivateResource either alone (lock and unlock) or in combination with with an IResourceLock whose constructor locks the resource and destructor unlocks the resource. If you only need the lock 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 creating the object, if necessary, is a mechanism of deferring creation of the object until it is needed.

IPrivateResource is a resource that is only used within the same process (as opposed to a public resource that is used by more than one process via its name).

You can use IPrivateResource either alone (lock and unlock) or in combination with an IResourceLock whose constructor locks the resource and destructor unlocks the resource. If you only need the lock for a block of code, use IResourceLock.

You can use IPrivateResource as a static key to serialize access to a private resource. You can also use the class as a mechanism to ensure that the static resource is constructed prior to being used.

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 directly using the static pointer.
- When the resource is accessed through the static function, allocate it if the static pointer is 0.

© Copyright IBM Corp. 1994

- Provide a class that represents the static pointers for a particular class or component.

Constructors

```
public:
IPrivateResource();
```

The only way to construct objects of this class is the default constructor, which does not accept any parameters.

Public Members

Figure 222. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
lock	IResource
operator <<	IBase*
operator <<	IVBase*
unlock	IResource

Note: * Application Support Class Library

Protected Members

handle

```
ISemaphoreHandle &handle();
```

Returns the handle for the resource.

Figure 223. Inherited Protected Members

Member function	Class
handle	IResource

Note: * Application Support Class Library

Chapter 152. IProcedureAddress

Derivation

IBase
  IProcedureAddress

Header File: iprocadr.hpp

The IProcedureAddress class provides a wrapper for OS/2 dynamic link library (.dll) entry points obtained using DosLoadModule or DosGetProcAddr. The library uses DosLoadModule to do a runtime link to a dynamic link library. The library uses DosGetProcAddr to acquire the runtime address of a function in a dynamic link library. When used together, they allow the calling of code that was not known to the linker when the executable module was built.

This class provides for:

- Constructing objects from entry point names (or ordinal entry point numbers) and dynamic link library objects (or .dll names).
- Calling the entry point without having to deal with casting to and from the generic "PFN" type supported by DosGetProcAddr.
- Managing, along with DynamicLinkLibrary (see page 244), the loading and freeing of the associated dynamic link library file.

Portability Considerations: AIX does not support dynamic link libraries.

Motif Information: AIX does not support this class or dynamic link libraries.

Presentation Manager Information: **Customization (Template Argument)**

IProcedureAddress is a template class that is instantiated with the following template argument:

PtrToFnType
Type of the entry point to be loaded or called; this must be a pointer-to-function type.

Constructors

Condition: (2) OS/2 Only

```
public:
IProcedureAddress(
   const char *entryPoint,
   const char *dllName);
```

Condition: (2) OS/2 Only

© Copyright IBM Corp. 1994

```
public:
IProcedureAddress(
   const char *entryPoint,
   IDynamicLinkLibrary &aDLL);
```

Condition: (2) OS/2 Only

```
public:
IProcedureAddress(
   unsigned long ordinal,
   const char *dllName);
```

Condition: (2) OS/2 Only

```
public:
IProcedureAddress(
   unsigned long ordinal,
   IDynamicLinkLibrary &aDLL);
```

You must provide two pieces of information to construct objects of this class:

- The entry point. You can specify either an entry point name (const char *) or entry point ordinal (unsigned).
- The dynamic link library from which to load the entry point. You can specify the .dll either as a reference to an existing object of IDynamicLinkLibrary (see page 244) or simply by the dynamic link library name.

The destructor of the object that is created by IProcedureAddress frees the dynamic link library. If you need to use the dynamic link library again before it is freed, construct the IProcedureAddress by using an explicitly created IDynamicLinkLibrary object instead of the dynamic link library name.

Public Members

is32Bit

Condition: (2) OS/2 Only

```
Boolean is32Bit() const;
```

Returns whether the entry point is 32-bit.

operator PtrToFnType

Condition: (2) OS/2 Only

```
operator PtrToFnType() const;
```

This operator is provided to permit instances of this template class to be used in the same way as pointers-to-functions, which is like regular function names.

Figure 224. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 153. IProcessId

Derivation

IBase
  IHandle
    IProcessId

Header File: ihandle.hpp

Objects of the IProcessId class access numeric identifiers for processes.

Motif Information: IProcessId is an alias for the system type pid_t.

Presentation Manager Information: IProcessId is an alias for the OS/2 Programmer's Toolkit type PID.

Constructors

```
public:
IProcessId(Value pid = 0);
```

You can construct objects of this class from a process ID (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IProcessId value.

Figure 225. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 154. IProfile

Derivation

IBase
  IVBase
    IProfile

Header File: iprofile.hpp

Objects of the IProfile class represent profile data sets. They provide functions to query and set persistent application data based on application-defined keys. The profile data set stores information using the following two keys:

- The first key is the application name, which is stored as a string.
- The second key is the key name, which is stored as either a string or an integer.

Profile data can be stored as text, binary and integer by the keys.

Motif Information: The objects representing profile data sets are Xrm database files.

The profile data set stores information using the following two keys:

- The first key is the application name, which is stored as a string.
- The second key is the key name, which is stored as a string.

Both of these names must be stored as strings with no imbedded spaces. You *cannot* store key names as integers. You can store the profile data itself as text and integer values using the keys.

Presentation Manager Information: The objects representing profile data sets are .INI files.

Note: The number of .INI files that the OS/2 Presentation Manager lets you open at one time varies depending on system limitations.

Nested Classes

IProfile::Cursor

Constructors

```
public:
IProfile(const char *profileName);

public:
IProfile(const IProfile &aProfile);
```

You can construct objects of this class using the name of the profile data set.

© Copyright IBM Corp. 1994

Public Members

addOrReplaceElementWithKey

```
virtual IProfile &addOrReplaceElementWithKey(
                    const char *key,
                    long data,
                    const char *applName = 0);

virtual IProfile &addOrReplaceElementWithKey(
                    const char *key,
                    const IString &data,
                    const char *applName = 0);
```

Writes the specified number, text, or binary data. You must specify text and binary data as an IString.

Motif Information: AIX does not support writing of binary data.

applicationOrKeyAt

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

Returns the application name or key at the specified cursor.

containsApplication

```
virtual Boolean containsApplication(
                const char *applName) const;
```

If the profile contains data for the specified application name, true is returned.

containsKeyName

```
virtual Boolean containsKeyName(
                const char *key,
                const char *applName = 0) const;
```

If the profile contains data for the specified application and key pair, true is returned.

defaultApplicationName

```
virtual IString defaultApplicationName() const;
```

Returns the default application name.

deleteElementWithApplication

```
virtual IProfile &deleteElementWithApplication(
                    const char *applName = 0);
```

Removes the data for all keys for the specified application name.

deleteElementWithKey

```
virtual IProfile &deleteElementWithKey(
                    const char *key,
                    const char *applName = 0);
```

Removes the data at a specified application and key pair.

elementWithKey

```
virtual IString elementWithKey(
                  const char *key,
                  const char *applName = 0) const;
```

Reads the application data and returns it as an IString.

handle

```
virtual IProfileHandle handle() const;
```

Returns the profile handle.

integerWithKey

```
virtual long integerWithKey(
               const char *key,
               const char *applName = 0) const;
```

Reads the application data and returns it as a long integer.

name

```
virtual IString name() const;
```

Returns the profile file name.

numberOfApplications

Condition: (1) Ignored on AIX

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

Returns the number of application names in the profile.

numberOfKeys

```
virtual unsigned long numberOfKeys(
                        const char *applName = 0) const;
```

Returns the number of keys for the application name.

operator =

```
virtual IProfile &operator =(const IProfile &aProfile);
```

Tracks references to the resource.

setDefaultApplicationName

```
virtual IProfile &setDefaultApplicationName(
                    const char *applName);
```

Sets the default application name. All subsequent calls with a NULL application name use this default name.

systemProfile

Condition: (2) OS/2 Only

```
static IProfile systemProfile();
```

Returns the system profile.

userProfile

Condition: (2) OS/2 Only

```
static IProfile userProfile();
```

Returns the user profile.

Figure 226. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 155. IProfile::Cursor

Derivation

IBase
  IVBase
    IProfile::Cursor

Header File: iprofile.hpp

Objects of the IProfile::Cursor class create and manage the cursor for an IProfile (see page 671) object. IProfile uses this cursor to traverse the application names or keys in a profile. 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 profile, one item at a time.

Constructors

```
public:
Cursor(IProfile &profile);

public:
Cursor(IProfile &profile, const char *applName);
```

You can construct objects of this class by providing one of the following:

- A reference to the profile. Use this constructor to create a cursor to iterate over the application names.
- A reference to the profile and an application name. Use this constructor to create a cursor to iterate over the keys.

Note: The cursor object is invalidated to the current state of application names or key names in the profile file.

Public Members

invalidate

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

Marks the cursor as invalid.

isValid

```
virtual Boolean isValid(Boolean checkFile = false) const;
```

If the cursor is not off the end, true is returned.

Motif Information: AIX does not support *checkFile*.

© Copyright IBM Corp. 1994

Presentation Manager Information: 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.

setToFirst

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

Sets the cursor's position to the first application or key.

setToLast

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

Sets the cursor's position to the last application or key.

setToNext

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

Sets the cursor's position to the next application or key.

setToPrevious

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

Set the cursor's position to the previous application or key.

Figure 227. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 156. IProfileHandle

Derivation

IBase
  IHandle
    IProfileHandle

Header File: ihandle.hpp

Objects of the IProfileHandle class access profiles.

Motif Information: IProfileHandle is an alias for the X/Motif types XrmHashBucket and XrmDatabase.

Presentation Manager Information: IProfileHandle is an alias for the OS/2 Programmer's Toolkit type HINI.

Constructors

```
public:
IProfileHandle(Value hini = 0);
```

Condition: (3) AIX only

```
public:
IProfileHandle(_XrmHashBucketRec *hini);
```

Condition: (3) AIX only

```
public:
IProfileHandle(int hini);
```

You can construct objects of this class from a profile handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

Condition: (2) OS/2 Only

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

Returns the IProfileHandle value.

© Copyright IBM Corp. 1994

operator _XrmHashBucketRec *

Condition: (3) AIX only

```
operator _XrmHashBucketRec *() const;
```

Returns the handle value as a native X-Toolkit handle object.

Figure 228. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Chapter 157. IProgressIndicator

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IProgressIndicator

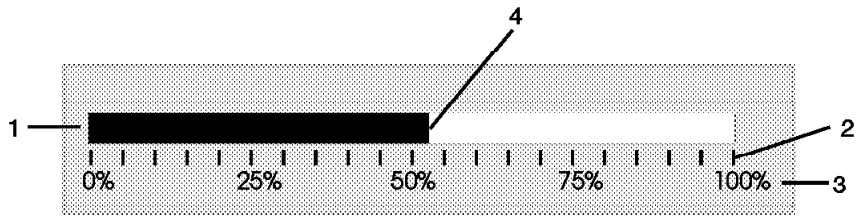
Header File: islider.hpp

Objects of the IProgressIndicator class are a read-only version of the slider control. ISlider (see page 792) describes the slider control.

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

- IFocusHandler (see page 307)
- IResizeHandler (see page 732)
- IPaintHandler (see page 647)
- IMouseClickHandler (see page 539)

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.

Note: The arm position is not set to the same tick position when the progress indicator control window is resized. Typically, this occurs when both of the following occur:

- The progress indicator is on a canvas.
- This constructor uses the default for tick spacing.

© Copyright IBM Corp. 1994

In such cases, the arm is positioned at the same pixel position, instead of the same relative position from home.

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. (OS/2 environment)
- The arm starts from the right edge.
- The indicator is vertical.

Motif Information: The IProgressIndicator constructor creates objects of this class using several Motif widgets:

- An XmDrawingArea widget is created with an XmScrollBar child if the ribbonStrip style is enabled or with an XmScale child, otherwise. IWindow::handle returns the handle of the XmDrawingArea widget. The XmDrawingArea widget also has one or two XmForm widget children.
- The XmForm widgets are used to 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.

Nested Classes

IProgressIndicator::Style

Constructors

```
public:
IProgressIndicator(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
```

```
   const IRectangle &initial,
   unsigned long numberOfTicks,
   unsigned long tickSpacing = 0,
   const Style &style = defaultStyle ( ));

public:
IProgressIndicator(
   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 ( ));
```

Condition: (2) OS/2 Only

```
public:
IProgressIndicator(unsigned long windowId, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
IProgressIndicator(const IWindowHandle &handle);

protected:
IProgressIndicator();
```

You can construct public objects of this class by:

- Using the primary constructor

  windowId A unique ID for the progress indicator control.

  parent The parent window.

  owner The owner window.

  initial (Optional) 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 (see page 717) constructor.

  numberOfTicks
  The number of ticks to place on the primary scale of the progress indicator.

  tickSpacing
  (Optional) 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.

  The tick spacing can also be changed dynamically by using the setTicks member function.

style (Optional) The style of the control.

- 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 (see page 942) or IWindow::moveSizeTo (see page 932). 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 size.

  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.

- Specifying a parent window on the ID of a progress indicator control
- Specifying a window handle

  handle The window handle of the progress indicator control.

In addition, the User Interface Class Library provides a protected constructor used by the ISlider (see page 792) class during construction.

Public Members

alignment

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

Returns the current alignment of this progress indicator object. The returned value is an enumerator provided by Alignment (see page 693).

armPixelOffset

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

Returns the offset, in pixels, of the arm from the home position.

armRange

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

Returns the number of pixels over which the arm can move.

armTickOffset

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

Returns the position of the arm as a tick number. Ticks are numbered starting at 0.

backgroundColor

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

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

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 695) unless you have changed the style using setDefaultStyle (see page 686).

disableDrawItem

Condition: (1) Ignored on AIX

```
virtual IProgressIndicator &disableDrawItem();
```

Sets the handleDrawItem (see page 695) style to off.

Motif Information: The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

disableRibbonStrip

```
virtual IProgressIndicator &disableRibbonStrip();
```

Sets the ribbonStrip (see page 695) style to off.

disableSnapToTick

```
virtual IProgressIndicator &disableSnapToTick();
```

Sets the snapToTickMark (see page 695) style to off.

enableDrawItem

Condition: (1) Ignored on AIX

```
virtual IProgressIndicator &enableDrawItem(
                              Boolean enableDrawItem = true);
```

Enables or disables the handleDrawItem (see page 695) style for the progress indicator.

Motif Information: The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

enableRibbonStrip

```
virtual IProgressIndicator &enableRibbonStrip(
                              Boolean enableRibbonStrip = true);
```

Enables or disables the ribbonStripOn (see page 695) style for the progress indicator.

enableSnapToTick

```
virtual IProgressIndicator &enableSnapToTick(
                              Boolean enableSnapToTick = true);
```

Enables or disables the snapToTickMark (see page 695) style for the progress indicator.

foregroundColor

Condition: (3) AIX only

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

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

homePosition

```
HomePosition homePosition() const;
```

Returns the current home position for this progress indicator object. The returned value is an enumerator provided by HomePosition (see page 693).

isDrawItemEnabled

Condition: (1) Ignored on AIX

```
Boolean isDrawItemEnabled() const;
```

If the handleDrawItem (see page 695) style is set, true is returned.

Motif Information: The AIX release does not support the handleDrawItem style. Therefore, this member function has no effect in AIX.

isRibbonStripEnabled

```
Boolean isRibbonStripEnabled() const;
```

If the ribbonStrip (see page 695) style is set, true is returned.

isSnapToTickEnabled

```
Boolean isSnapToTickEnabled() const;
```

If the snapToTickMark (see page 695) style is set, true is returned.

isVertical

```
Boolean isVertical() const;
```

If the progress indicator is oriented vertically, true is returned. If it is oriented horizontally, false is returned.

moveArmToPixel

```
virtual IProgressIndicator &moveArmToPixel(
                              unsigned long armOffset);
```

Moves the arm to a pixel offset relative to the home position.

armOffset Number of pixels to offset the arm.

moveArmToTick

```
virtual IProgressIndicator &moveArmToTick(
                              unsigned long tickNumber);
```

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.

moveSizeTo

Condition: (3) AIX only

```
virtual IProgressIndicator &moveSizeTo(
                              const IRectangle &rectangle);
```

Changes the position and size of the progress indicator window using a lower-left origin coordinate system.

aRectangle
A rectangle for the progress indicator control.

numberOfTicks

```
unsigned long numberOfTicks(Scale scale) const;
```

Returns the number of ticks for the specified scale.

primaryScale

```
Scale primaryScale() const;
```

Returns the current primary scale of this progress indicator object. The returned value is an enumerator provided by Scale (see page 693).

setBackgroundColor

Condition: (3) AIX only

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

Sets the background color to the specified color.

setDefaultStyle

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

Sets the default style for all subsequent progress indicators.

style Use the styles provided by IProgressIndicator::Styles (see page 695) to specify the default style.

setForegroundColor

Condition: (3) AIX only

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

Sets the color for the ticks and label text.

setHomePosition

```
virtual IProgressIndicator &setHomePosition(
                              HomePosition home = homeBottomLeft);
```

Sets the home position of the progress indicator.

home Enumerator value for the home position. Use the enumeration HomePosition (see page 693) to specify the home position.

setPrimaryScale

```
virtual IProgressIndicator &setPrimaryScale(
                              Scale primaryScale = scale1);
```

Sets the primary scale of the progress indicator.

primaryScale
Use the enumeration Scale (see page 693) to specify which scale to set.

setShaftBreadth

```
virtual IProgressIndicator &setShaftBreadth(
                              unsigned long breadth);
```

Sets the shaft width in pixels for vertical progress indicators. Sets the shaft height in pixels for horizontal progress indicators.

breadth Number of pixels.

setShaftPosition

Condition: (1) Ignored on AIX

```
virtual IProgressIndicator &setShaftPosition(
                              const IPoint &lowerLeft);
```

Sets the position, in pixels, of the lower-left point of the shaft. The position of the shaft is relative to the lower-left corner of the progress indicator window.

Motif Information: The AIX release ignores this function.

setTickLength

```
virtual IProgressIndicator &setTickLength(
                              unsigned long tickNumber,
                              unsigned long length);

virtual IProgressIndicator &setTickLength(
                              unsigned long length);
```

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.

Motif Information: 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.

setTicks

```
virtual IProgressIndicator &setTicks(
                              Scale scale,
                              unsigned long numberOfTicks,
                              unsigned long tickSpacing = 0);

virtual IProgressIndicator &setTicks(
                              unsigned long scale1NumberOfTicks,
                              unsigned long scale2NumberOfTicks,
                              unsigned long scale1TickSpacing = 0,
                              unsigned long scale2TickSpacing = 0);
```

For both scale1 and scale2, this function sets the following:

- Number of ticks
- Number of pixels between ticks

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.

numberOfTicks
The number of ticks to place on scale of the progress indicator.

tickSpacing
The number of pixels between ticks on the scale.

setTickText

```
virtual IProgressIndicator &setTickText(
                              unsigned long tickNumber,
                              const char *text);

virtual IProgressIndicator &setTickText(
                              unsigned long tickNumber,
                              const IResourceId &text);
```

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.

shaftPosition

```
virtual IPoint shaftPosition() const;
```

Returns the lower-left point of the shaft relative to the lower-left corner of the progress indicator window.

shaftSize

```
virtual ISize shaftSize() const;
```

Returns the size of the shaft in pixels.

tickLength

```
unsigned long tickLength(unsigned long tickNumber) const;
```

Returns the length, in pixels, of the tick at the specified index.

tickNumber
A 0-based tick number relative to the home position.

tickPosition

```
IPoint tickPosition(unsigned long tickNumber) const;
```

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.

tickSpacing

```
unsigned long tickSpacing(Scale scale) const;
```

Returns the number of pixels between ticks for the specified scale.

scale Use the enumeration Scale (see page 693) to specify for which scale tick spacing is set.

tickText

```
IString tickText(unsigned long tickNumber) const;
```

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.

Figure 229. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow

Figure 229. Inherited Public Members

Member function	Class
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow

Figure 229. Inherited Public Members

Member function	Class
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow

Figure 229. Inherited Public Members

Member function	Class
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow

Figure 229. Inherited Public Members

Member function	Class
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns an ISize (see page 790) indicating the minimum size of the progress indicator. ICanvas (see page 63) and its derived classes use this function.

initialize

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

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.

registerCallbacks

Condition: (3) AIX only

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

Called during construction of the IProgressIndicator window.

This function registers all possible callbacks and X event handlers to this IProgressIndicator (see page 924) for events it receives. IHandler (see page 402) 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.

unregisterCallbacks

Condition: (3) AIX only

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

Removes all callbacks and X event handlers from this IWindow (see page 924) that were added by registerCallbacks (see page 691). 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.

Figure 230. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow

Figure 230. Inherited Protected Members

Member function	Class
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

```
public:
enum 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

```
public:
enum HomePosition
  {
  homeTopRight,
  homeBottomLeft,
  };
```

Use these enumerators to specify the home position of the progress indicator:

homeBottomLeft
Home position is at the bottom for a vertical progress indicator and at the left for a horizontal progress indicator.

homeTopRight
Home position is at the top for a vertical progress indicator and at the right for a horizontal progress indicator.

Scale

```
public:
enum 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.

Chapter 158. IProgressIndicator::Style

Derivation

IBase
  IBitFlag
    IProgressIndicator::Style

Header File: islider.hpp

The nested class IProgressIndicator::Style provides a set of valid progress indicator styles for the static member functions of the class IProgressIndicator (see page 679). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

alignBottom
Places the progress indicator at the bottom of the window.

alignCentered
Centers the progress indicator in the window.

alignLeft
Places the progress indicator at the left of the window.

alignRight
Places the progress indicator at the right of the window.

alignTop
Places the progress indicator at the top of the window.

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible | horizontal | alignCentered | homeLeft | ribbonStrip | primaryScale1.

handleDrawItem
Specifies that an IDrawItemEvent be dispatched to the progress indicator control whenever the shaft, ribbon strip, arm, and background are drawn.

homeBottom
Specifies that the progress indicator's arm and scale start at the bottom.

homeLeft
Specifies that the progress indicator's arm and scale start at the left.

homeRight
Specifies that the progress indicator's arm and scale start at the right.

© Copyright IBM Corp. 1994

homeTop
Specifies that the progress indicator's arm and scale start at the top.

horizontal
Specifies that the progress indicator be built horizontally.

primaryScale1
Uses the progress indicator's scale 1 for positioning the arm, which is located above or right of the progress indicator.

primaryScale2
Uses the progress indicator's scale 2 for positioning the arm, which is located below or left of the progress indicator.

ribbonStrip
Fills the progress indicator's shaft with color between the arm and home position when the arm is moved.

snapToTickMark
Specifies that the progress indicator's arm snap to the nearest tick when moved between two ticks.

vertical
Specifies that the progress indicator be built vertically.

Motif Information: AIX does not support the handleDrawItem style.

Public Members

Figure 231. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 232. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 159. IPushButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              IPushButton

Header File: ipushbut.hpp

Objects of the IPushButton class create and manage push button control windows.

The standard push button generates an ICommandEvent (see page 128). However, the application can change the window style value to generate a help event or system command event.

To change the push button event (message) processing, call:

- IPushButton::enableHelp (see page 699)
- IPushButton::enableSystemCommand (see page 700)

These functions add or remove the styles:

- help (see page 705)
- systemCommand (see page 705)

When the user selects a push button, the push button generates an ICommandEvent that is routed to its owner window, or a help request. You can process an ICommandEvent with an ICommandHandler.

If the application does not create the push button from a dialog template, the application is responsible for processing the **Enter** key and calling the push button's inherited function IButton::click (see page 56).

You can derive classes from the following handlers and attach them to a push button control:

- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)

Motif Information: The AIX release of the User Interface Class Library does not support dialog templates.

© Copyright IBM Corp. 1994

Nested Classes

IPushButton::Style

Constructors

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

public:
IPushButton(unsigned long id, IWindow *parentDialog);

public:
IPushButton(const IWindowHandle &handle);
```

Condition: (3) AIX only

```
protected:
IPushButton();
```

You can construct objects of this class by:

- Creating the specified push button control and an object for it.

  id A push button control ID.

  parent The parent window.

  owner The owner window.

  initial (Optional) The initial position and size of the control you are constructing.

  style (Optional) The control's characteristics.

- Creating the object for the specified push button control.

  parentDialog
  The parent window.

- Creating the object for the specified push button control.

  handle The window handle of an existing push button control.

Public Members

addBorder

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

Adds or removes the push button border.

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

Returns the default style. The default style is classDefaultStyle (see page 705) unless you have changed the style using setDefaultStyle (see page 700).

disableBorder
```
virtual IPushButton &disableBorder();
```

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.

disableDefault
```
virtual IPushButton &disableDefault();
```

Removes the style defaultButton (see page 705) from the push button.

disableHelp
```
virtual IPushButton &disableHelp();
```

Removes the style help (see page 705) from the push button.

disableSystemCommand
```
virtual IPushButton &disableSystemCommand();
```

Removes the style systemCommand (see page 705) from the push button.

enableBorder
```
virtual IPushButton &enableBorder(Boolean add = true);
```

Adds or removes the push button border.

enableDefault
```
virtual IPushButton &enableDefault(Boolean enable = true);
```

Adds or removes the style defaultButton (see page 705).

enableHelp
```
virtual IPushButton &enableHelp(Boolean enable = true);
```

Adds or removes the style help (see page 705).

enableSystemCommand

```
virtual IPushButton &enableSystemCommand(
                        Boolean enable = true);
```

Adds or removes the style systemCommand (see page 705).

isBorder

```
Boolean isBorder() const;
```

If the push button has a border, true is returned.

isDefault

```
Boolean isDefault() const;
```

If the style defaultButton (see page 705) is set, true is returned.

isHelp

```
Boolean isHelp() const;
```

If the style help (see page 705) is set, true is returned.

isSystemCommand

```
Boolean isSystemCommand() const;
```

If the style systemCommand (see page 705) is set, true is returned.

removeBorder

```
virtual IPushButton &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.

setDefaultStyle

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

Sets the default style for all subsequent push buttons.

style Use the styles provided by IPushButton::Styles (see page 705) to specify the default style.

Figure 233. Inherited Public Members

Member function	Class
activeColor	IWindow
allowsMouseClickFocus	IButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IButton
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
click	IButton
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disabledForegroundColor	IButton
disableGroup	IControl
disableMouseClickFocus	IButton
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableMouseClickFocus	IButton
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IButton
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow

Figure 233. Inherited Public Members

Member function	Class
hideSourceEmphasis	IWindow
highlight	IButton
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IButton
hiliteForegroundColor	IWindow
hiliteForegroundColor	IButton
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHighlighted	IButton
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow

Figure 233. Inherited Public Members

Member function	Class
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl

Figure 233. Inherited Public Members

Member function	Class
setText	IButton
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
unhighlight	IButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size this push button control can be based on the text string length and the current font.

passEventToOwner

Condition: (3) AIX only

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

Determines if the event is passed on to the owner.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

unregisterCallbacks

Condition: (3) AIX only

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

Removes any callbacks and X event handlers added by the registerCallbacks function.

Figure 234. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier

Figure 234. Inherited Protected Members

Member function	Class
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow

Figure 234. Inherited Protected Members

Member function	Class
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 160. IPushButton::Style

Derivation

IBase
  IBitFlag
    IPushButton::Style

Header File: ipushbut.hpp

The nested class IPushButton::Style provides a set of valid push button styles for the static member functions of the class IPushButton (see page 697). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IButton::Style (see page 61)

Note: For this release, you must write your own user-button class. You can derive icon buttons objects from the IGraphicPushButton class.

Styles

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible.

defaultButton
Specifies the push button default. If the parent or owner window is a canvas or frame window, the **Enter** key can select the push button without having the cursor on the push button. Visually, a default push button has a darker and thicker border.

help
Generates a help request, instead of a command event, when the push button is selected.

noBorder
Displays the push button without a border.

systemCommand
Generates a system command event, instead of a command event, when the push button is selected. The only portable system command is ISystemMenu::idClose. This is the system command for closing a window.

Motif Information: For Motif, the general help window for the owner window is displayed when the **Help** push button is selected. This is in accordance with the *Motif Style Guide* for dialogs.

© Copyright IBM Corp. 1994

Public Members

Figure 235. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 236. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 161. IRadioButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              ISettingButton
                IRadioButton

Header File: iradiobt.hpp

Objects of the IRadioButton class create and manage 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 (see page 764) and adding your handler either to the radio button or to its owner window.

Motif Information: The IRadioButton constructor creates objects of this class using the XmToggleButton widget. IWindow::handle (see page 928) 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.

Nested Classes

IRadioButton::Style

Constructors

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

© Copyright IBM Corp. 1994

```
public:
IRadioButton(unsigned long id, IWindow *parentDialog);

public:
IRadioButton(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Creating the specified radio button control and an object for it
- Creating the object for the specified radio button control
- Creating the object for the specified radio button control using a handle

Public Members

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 713) unless you have changed the style using setDefaultStyle (see page 709).

disableAutoSelect

Condition: (1) Ignored on AIX

```
virtual IRadioButton &disableAutoSelect();
```

Removes the style autoSelect (see page 713) from the radio button control.

disableCursorSelect

Condition: (1) Ignored on AIX

```
virtual IRadioButton &disableCursorSelect();
```

Adds the radio button style noCursorSelect (see page 713).

enableAutoSelect

Condition: (1) Ignored on AIX

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

Adds or removes the style autoSelect (see page 713).

Motif Information: The autoSelect style is always in effect.

enableCursorSelect

Condition: (1) Ignored on AIX

```
virtual IRadioButton &enableCursorSelect(
                        Boolean enable = true);
```

Makes the radio button cursor-selectable.

isAutoSelect

Condition: (1) Ignored on AIX

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

If the radio button control has the style autoSelect (see page 713) set, true is returned. Otherwise, false is returned.

Motif Information: AIX always uses autoSelect and returns true.

isCursorSelect

Condition: (1) Ignored on AIX

```
Boolean isCursorSelect() const;
```

If the radio button is cursor-selectable, true is returned.

Motif Information: This function returns false.

selectedIndex

```
unsigned long selectedIndex() const;
```

Returns the 0-based index of the selected radio button in a group. If no radio button is selected, -1 is returned.

Presentation Manager Information: In OS/2, the index of any control in the group can be returned.

Warning: If you call selectedIndex without a radio button being selected, some versions of OS/2 might experience system instability.

setDefaultStyle

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

Sets the default style for all subsequent radio buttons.

style Use the styles provided by IRadioButton::Styles (see page 713) to specify the default style.

Figure 237. Inherited Public Members

Member function	Class
activeColor	IWindow
allowsMouseClickFocus	IButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IButton
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
click	IButton
defaultPushButton	IWindow
deselect	ISettingButton
disable	IWindow
disableAutoSelect	ISettingButton
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disabledForegroundColor	IButton
disableGroup	IControl
disableMouseClickFocus	IButton
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableAutoSelect	ISettingButton
enableGroup	IControl
enableMouseClickFocus	IButton
enableNotification	INotifier
enableNotification	IWindow
enableNotification	ISettingButton
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IButton

Figure 237. Inherited Public Members

Member function	Class
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
highlight	IButton
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IButton
hiliteForegroundColor	IWindow
hiliteForegroundColor	IButton
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isAutoSelect	ISettingButton
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHighlighted	IButton
isLayoutDistorted	IWindow
isSelected	ISettingButton
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*

Figure 237. Inherited Public Members

Member function	Class
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
select	ISettingButton
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow

Figure 237. Inherited Public Members

Member function	Class
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IButton
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
unhighlight	IButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the minimum size that this radio button control based on the text string length and the current font.

registerCallbacks

Condition: (3) AIX only

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

Registers Motif callbacks for the widget created during object construction of this class.

unregisterCallbacks

Condition: (3) AIX only

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

Unregisters Motif callbacks for the widget created during object construction of this class.

Figure 238. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ISettingButton
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow

Figure 238. Inherited Protected Members

Member function	Class
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 162. IRadioButton::Style

Derivation

IBase
  IBitFlag
    IRadioButton::Style

Header File: iradiobt.hpp

The nested class IRadioButton::Style provides a set of valid radio button styles for the static member functions of the class IRadioButton (see page 707). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IButton::Style (see page 61)

Styles

autoSelect
Specifies a selection technique in which the user changes the current selection automatically by moving the keyboard cursor.

classDefaultStyle
Specifies the original default style for this class, which is autoSelect | IWindow::visible.

noCursorSelect
Prevents the radio button from selecting itself when given the focus as a result of an arrow key or tab key.

Motif Information: The AIX version does not support the noCursorSelect and autoSelect styles.

Public Members

Figure 239. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*

Figure 239. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 240. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 163. IRange

Derivation

IBase
  IPair
    IRange

Header File: ipoint.hpp

Objects of the IRange class represent a range of IPair::Coord values between a specified lower and upper bound (inclusive).

Constructors

```
public:
IRange();

public:
IRange(const IPair &aPair);

public:
IRange(Coord lower, Coord upper);
```

You can construct objects of this class in the following ways:

- By using the default constructor. The resulting range is [0, 0].
- From any ordered pair (a, b). The resulting range is [a, b].
- From any specified lower and upper bounds.

Public Members

includes

```
Boolean includes(Coord aValue) const;
```

Returns true if the range contains the specified coordinate value.

lowerBound

```
Coord lowerBound() const;
```

Returns the lower bound of the range.

setLowerBound

```
IRange &setLowerBound(Coord lower);
```

Sets the lower bound of the range.

© Copyright IBM Corp. 1994

setUpperBound

```
IRange &setUpperBound(Coord upper);
```

Sets the upper bound of the range.

upperBound

```
Coord upperBound() const;
```

Returns the upper bound of the range.

Figure 241. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IPair
asString	IBase*
asString	IPair
coord1	IPair
coord2	IPair
distanceFrom	IPair
dotProduct	IPair
maximum	IPair
minimum	IPair
operator !=	IPair
operator %	IPair
operator %=	IPair
operator *	IPair
operator *=	IPair
operator +	IPair
operator +=	IPair
operator -	IPair
operator -=	IPair
operator /	IPair
operator /=	IPair
operator <	IPair
operator <<	IBase*
operator <<	IPair
operator <=	IPair
operator ==	IPair
operator >	IPair
operator >=	IPair
scaleBy	IPair
scaledBy	IPair
setCoord1	IPair
setCoord2	IPair

Figure 241. Inherited Public Members

Member function	Class
transpose	IPair

Note: * Application Support Class Library

Chapter 164. IRectangle

Derivation

IBase
  IRectangle

Header File: irect.hpp

Objects of the IRectangle class represent a rectangular area defined by two points that form opposite corners of the rectangle. The lower left (origin) and upper right (corner) corners represent the rectangle.

Mathematically, a rectangle includes all the points on the lines that form its lower and left edges. It does not include the points that lie on either its upper or right edges.

Various graphics and windowing classes, as well as their member functions, use IRectangles.

Presentation Manager Information: You can also construct objects of this class by providing a Presentation Manager Toolkit RECTL structure.

Constructors

```
public:
IRectangle();

public:
IRectangle(const IPoint &pt1, const IPoint &pt2);

public:
IRectangle(const IPoint &pt, const ISize &size);

public:
IRectangle(
   Coord left,
   Coord bottom,
   Coord right,
   Coord top);
```

Condition: (2) OS/2 Only

```
public:
IRectangle(const struct _RECTL &rectl);

public:
IRectangle(const struct tagRECT &rectl);

public:
IRectangle(Coord width, Coord height);
```

© Copyright IBM Corp. 1994

```
public:
IRectangle(const IPair &aPair);
```

You can construct objects of this class by providing:

- A NULL rectangle (0,0) x (0,0). This is the default constructor.
- Two specified points at opposite corners.
- A specified point (presumed to be the lower-left corner) and a specified size.
- Four specified values representing the coordinates for two points at opposite corners.
- A rectangle with opposite corners at (0,0) and (width, height). You can specify the width and height either as two (long) integers or as an IPair (see page 649), ordered pair of coordinates.

Note: The library constructs rectangles by giving two points that are taken as opposite corners, not as origin and corner (or lower-left and upper-right corners). This ensures that internally the origin and corner points always are such that the origin is less than or equal to the corner.

Public Members

area

```
Coord area() const;
```

Returns the area of the rectangle.

asDebugInfo

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

Renders the rectangle as a diagnostic representation.

asRECTL

Condition: (2) OS/2 Only

```
struct _RECTL asRECTL() const;
```

Renders the rectangle as a Presentation Manager Toolkit RECTL structure.

asString

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

Renders the rectangle as an IString("IRectangle(x1,y1,x2,y2)").

bottom

```
Coord bottom() const;
```

Returns the Y-coordinate of the horizontal line that forms the bottom of the rectangle.

bottomCenter
```
IPoint bottomCenter() const;
```

Returns the X- and Y-coordinates of the bottom-center point of the rectangle.

bottomLeft
```
IPoint bottomLeft() const;
```

Returns the X- and Y-coordinates of the bottom-left corner of the rectangle.

bottomRight
```
IPoint bottomRight() const;
```

Returns the X- and Y-coordinates of the bottom-right corner of the rectangle.

center
```
IPoint center() const;
```

Returns the X- and Y-coordinates of the center point of the rectangle.

centerAt
```
IRectangle &centerAt(const IPoint &aPoint);
```

Moves the rectangle so that its center is at the specified point.

centeredAt
```
IRectangle centeredAt(const IPoint &aPoint) const;
```

Same as IRectangle::centerAt (see page 719), but returns a new rectangle, leaving the original unmodified.

contains
```
Boolean contains(const IRectangle &aRect) const;

Boolean contains(const IPoint &aPoint) const;
```

If the rectangle contains the specified point or rectangle, true is returned.

expandBy
```
IRectangle &expandBy(const IPair &aPair);

IRectangle &expandBy(Coord coord);
```

Moves the corners of the rectangle outward from the center by the specified amount. The specified amount can be either a scalar (long integer) or a point.

expandedBy
```
IRectangle expandedBy(Coord coord) const;

IRectangle expandedBy(const IPair &aPair) const;
```

Same as IRectangle::expandBy (see page 719), but returns a new rectangle, leaving the original unmodified.

height
```
Coord height() const;
```

Returns the height of the rectangle.

intersects
```
Boolean intersects(const IRectangle &aRect) const;
```

If the rectangle and specified rectangle overlap, true is returned.

left
```
Coord left() const;
```

Returns the X-coordinate of the vertical line that forms the left side of the rectangle.

leftCenter
```
IPoint leftCenter() const;
```

Returns the X- and Y-coordinates of the left-center point of the rectangle.

moveBy
```
IRectangle &moveBy(const IPair &aPair);
```

Moves the rectangle by the amount specified by *aPair*.

movedBy
```
IRectangle movedBy(const IPair &aPair) const;
```

Same as IRectangle::moveBy (see page 720), but returns a new rectangle, leaving the original unmodified.

movedTo
```
IRectangle movedTo(const IPoint &aPoint) const;
```

Same as IRectangle::moveTo (see page 720), but returns a new rectangle, leaving the original unmodified.

moveTo
```
IRectangle &moveTo(const IPoint &aPoint);
```

Moves the rectangle so that its bottom-left corner is at the specified point.

operator !=

```
Boolean operator !=(const IRectangle &aRect) const;
```

If the rectangles differ, true is returned.

operator &

```
IRectangle operator &(const IRectangle &aRect) const;
```

Returns a rectangle representing the intersection of the specified rectangles.

operator &=

```
IRectangle &operator &=(const IRectangle &aRect);
```

Resets the rectangle to its intersection with the specified rectangle.

operator <<

```
friend ostream &operator <<(
                  ostream &aStream,
                  const IRectangle &aRectangle);
```

Lets you write rectangles to ostreams in the conventional manner.

operator ==

```
Boolean operator ==(const IRectangle &aRect) const;
```

If the two rectangles are identical, true is returned.

operator |

```
IRectangle operator |(const IRectangle &aRect) const;
```

Returns the rectangle representing the union of the specified rectangles. This is the smallest rectangle that encompasses both specified rectangles.

operator |=

```
IRectangle &operator |=(const IRectangle &aRect);
```

Resets the rectangle to its union with the specified rectangle.

right

```
Coord right() const;
```

Returns the X-coordinate of the vertical line that forms the right side of the rectangle.

rightCenter

```
IPoint rightCenter() const;
```

Returns the X- and Y-coordinates of the right-center point of the rectangle.

scaleBy

```
IRectangle &scaleBy(double factor);

IRectangle &scaleBy(const IPair &aPair);

IRectangle &scaleBy(Coord coord);

IRectangle &scaleBy(double xfact, double yfact);
```

Scales the rectangle by the specified amount which can be one of the following:

- A scalar, either a long integer or double.
- A point that scales by different amounts in the X- and Y-axis directions.
- A pair of doubles. The function uses the first double to scale in the X-axis direction, the second in the Y-axis direction.

scaledBy

```
IRectangle scaledBy(double xfact, double yfact) const;

IRectangle scaledBy(const IPair &aPair) const;

IRectangle scaledBy(Coord coord) const;

IRectangle scaledBy(double factor) const;
```

Same as IRectangle::scaleBy (see page 722), but returns a new rectangle, leaving the original unmodified.

shrinkBy

```
IRectangle &shrinkBy(Coord coord);

IRectangle &shrinkBy(const IPair &aPair);
```

Moves the corners of the rectangle inward toward the center by the specified amount, either a scalar or a point.

Note: shrinkBy(anAmount) is always equivalent to expandBy(- anAmount), and vice versa.

shrunkBy

```
IRectangle shrunkBy(const IPair &aPair) const;

IRectangle shrunkBy(Coord coord) const;
```

Same as IRectangle::shrinkBy (see page 722), but returns a new rectangle, leaving the original unmodified.

size

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

Returns the ISize(width, height).

sizeBy

```
IRectangle &sizeBy(double xfact, double yfact);

IRectangle &sizeBy(const IPair &aPair);

IRectangle &sizeBy(Coord factor);

IRectangle &sizeBy(double factor);
```

Scales the rectangle by the specified value, leaving the rectangle at the same location because the bottom-left point remains fixed. The scaling factor is one of the following:

- An integer scalar, to scale by the same factor in both the X-axis and Y-axis directions
- A pair of integer scalars, to scale by different factors in the X-axis and Y-axis directions
- One or two floating point values, the function uses the two values to scale by different amounts in each direction.

sizedBy

```
IRectangle sizedBy(double factor) const;

IRectangle sizedBy(const IPair &aPair) const;

IRectangle sizedBy(Coord factor) const;

IRectangle sizedBy(double xfact, double yfact) const;
```

Same as IRectangle::sizeBy (see page 723), but returns a new rectangle, leaving the original unmodified.

sizedTo

```
IRectangle sizedTo(const IPair &aPair) const;
```

Same as IRectangle::sizeTo (see page 723), but returns a new rectangle, leaving the original unmodified.

sizeTo

```
IRectangle &sizeTo(const IPair &aPair);
```

Sizes the rectangle to the specified size.

top

```
Coord top() const;
```

Returns the Y-coordinate of the horizontal line that forms the top of the rectangle.

topCenter

```
IPoint topCenter() const;
```

Returns the X- and Y-coordinates of the top-center point of the rectangle.

topLeft
```
IPoint topLeft() const;
```

Returns the X- and Y-coordinates of the top-left corner of the rectangle.

topRight
```
IPoint topRight() const;
```

Returns the X- and Y-coordinates of the top-right corner of the rectangle.

width
```
Coord width() const;
```

Returns the width of the rectangle.

Figure 242. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Protected Members

validate
```
IRectangle &validate();
```

Corrects an invalid rectangle after creation, expansion, or intersection.

Chapter 165. IRefCounted

Derivation

IBase
  IVBase
    IRefCounted

Header File: irefcnt.hpp

The IRefCounted class is a public base class for any class that is reference counted. Such inheritance conveys the functional characteristics of maintaining a count of all references to the object and deferring destruction until all such references are destroyed.

By necessity, you can only allocate objects of this class in free store. The library enforces this by making the destructor for this class protected. As a result, the library only allows IRefCounted::removeRef (see page 726) and derived class destructors to call IRefCounted::IRefCounted. Derived classes should make their destructors protected also.

Typically, you use this class in conjunction with the corresponding IReference<T> (see page 727), where T is a derived class of IRefCounted.

Constructors

```
protected:
IRefCounted();
```

The constructor initializes the use count to 1. This presumes the newly constructed object's address is immediately used by some reference (or functionally equivalent) object. The object that created the IRefCounted object should not call IRefCounted::addRef (see page 725) because the constructor has already marked it as referenced.

The destructor is protected to disable creation of non-free store objects.

Public Members

addRef

```
virtual void addRef();
```

Adds a reference to the referred-to object.

© Copyright IBM Corp. 1994

removeRef

```
virtual void removeRef();
```

Removes a reference to the referred-to object. When the reference count goes to 0, this function deletes the referred-to object.

useCount

```
unsigned useCount() const;
```

Returns the use count for the referred-to object.

Figure 243. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 166. IReference

Derivation

IBase
  IReference

Header File: irefcnt.hpp

The template class IReference is derived from classes that serve as references. Objects of such classes serve as smart pointers to objects of the referenced class. Creating objects of this class increments the use count of the referenced object. Destruction of the object causes the use count of the referenced object to be decremented.

Typically, this class is referenced explicitly only as a public base class of the class that provides the additional capability of the reference class. For example:

```
class Foo { .. };
class FooRef : public IReference<Foo> {
// Additional FooRef functions...
};
```

The reference-counted class provided as the template argument is derived from the class IRefCounted (see page 725). It must have the member functions IRefCounted::addRef (see page 725) and IRefCounted::removeRef (see page 726) with equivalent semantics.

To construct an IReference, you must provide a pointer to an object of the referenced (reference-counted) class. All constructors of the real reference class (derived from IReference<T>) must provide such a pointer. Otherwise, the reference class has no additional responsibilities.

Notes:

1. The semantics of such reference or referent classes can have subtle complexities. The reference or the referent might behave in an extraordinary fashion.
2. A class can also serve as a reference by having as a data member and instance of class IReference<T>.
3. All members of the IReference class are public in order to permit the usage described in item 2.

Customization (Template Argument)

IReference is a template class that is instantiated with the following template argument:

T Specifies the name of the class of objects to which instances of the template class refer.

© Copyright IBM Corp. 1994

Constructors

```
public:
IReference(T *p = 0) : referent( p )

public:
IReference(const IReference < T > &source) :
   referent( source.referent )
```

You can construct objects of this class by using:

- The primary constructor which accepts a pointer to an instance of the referenced class. This also serves as the default constructor (defaulting the pointer parameter to 0).
- The copy constructor which the library provides to ensure that the reference counts for both the source and target referents are maintained properly.

The destructor ensures that the referenced object is de-referenced.

Public Members

operator *

```
T &operator *() const;
```

Pointer de-reference operator that provides access to the referenced object.

operator ->

```
T *operator ->() const;
```

Pointer operator that provides access to the referenced object.

operator =

```
IReference < T > &operator =(
                    const IReference < T > &source);

IReference < T > &operator =(T *p);
```

The assignment operator. You can assign one IReference to another or you can assign a pointer to the referenced type.

operator T *

```
operator T *() const;
```

Returns the referent.

Figure 244. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*

Figure 244. Inherited Public Members

Member function	Class
operator <<	IBase*

Note: * Application Support Class Library

Chapter 167. IRegionHandle

Derivation

IBase
  IHandle
    IRegionHandle

Header File: ihandle.hpp

Objects of the IRegionHandle class access a window's update region.

Motif Information: The AIX release does not support this class.

Presentation Manager Information: IRegionHandle is an alias for the OS/2 Programmer's Toolkit HRGN typedef.

Constructors

```
public:
IRegionHandle(Value hregion = 0);
```

You can construct objects of this class from an enumeration handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IRegionHandle value.

Figure 245. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 168. IResizeEvent

Derivation

IBase
  IVBase
    IEvent
      IResizeEvent

Header File: isizeevt.hpp

Objects of the IResizeEvent class represent an event routed to a window notifying it of sizing changes (after the window's size changes). The User Interface Class Library provides functions to return the previous and new window sizes.

Motif Information: The AIX release of the User Interface Class Library maps this event to the ConfigureNotify event.

Constructors

```
public:
IResizeEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IResizeHandler::dispatchHandlerEvent (see page 733) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

newSize

```
ISize newSize() const;
```

Returns the size of the window after it is resized.

oldSize

```
ISize oldSize() const;
```

Returns the size of the window before it is resized.

Figure 246. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*

Figure 246. Inherited Public Members

Member function	Class
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent

© Copyright IBM Corp. 1994

Figure 246. Inherited Public Members

Member function	Class
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 169. IResizeHandler

Derivation

IBase
  IVBase
    IHandler
      IResizeHandler

Header File: isizehdr.hpp

Objects of the IResizeHandler class handle the notification of resizing events. These events notify a window or control that it has grown or shrunk 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 (see page 403) to pass the window or control to the resize handler.

When the resize handler receives a resize event, it creates an object of IResizeEvent (see page 730) and routes that object to the virtual function IResizeHandler::windowResize (see page 733). 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 (see page 946) to process the resize event.

Constructors

```
protected:
IResizeHandler();
```

The only way to create objects of this class is from a derived class. To enforce this, the only constructors we provide for this class are protected.

© Copyright IBM Corp. 1994

Public Members

Figure 247. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a resize event is received, the appropriate virtual function is called.

windowResize

```
virtual Boolean windowResize(IResizeEvent &event) = 0;
```

Implemented by a derived class to handle a resizing event.

Figure 248. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 170. IResource

Derivation

IBase
  IVBase
    IResource

Header File: ireslock.hpp

The IResource class is the virtual 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 IPrivateResource (see page 666), or they can be shared across multiple processes as described by ISharedResource (see page 786).

Constructors

```
public:
IResource();
```

You cannot create objects of this class.

Public Members

lock

```
virtual IResource &lock(long timeOut = -1);
```

Acquires serialized access to this resource. You can specify the value for *timeout* in milliseconds with -1 indicating an indefinite wait and 0 requesting an immediate return.

Motif Information: The X timer services used to implement the resource lock timeout have a resolution of one second. The lock function rounds the timeout value specified up to the next one second interval.

unlock

```
virtual IResource &unlock();
```

Releases the access to this resource.

Figure 249. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*

Figure 249. Inherited Public Members

Member function	Class
operator <<	IVBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

handle

```
virtual ISemaphoreHandle &handle() = 0;
```

Returns the handle for the resource.

Chapter 171. IResourceId

Derivation

IBase
  IResourceId

Header File: ireslib.hpp

The IResourceId class is a wrapper class for a resource identifier. Part of the information contained in the class is the location of the resources that are described using either an IResourceLibrary (see page 738) or an IDynamicLinkLibrary (see page 244). Most classes in the User Interface Class Library typically accept an IResourceId as input to describe resources and provide the resource loading themselves.

Constructors

```
public:
IResourceId(
   unsigned long resourceId,
   const IResourceLibrary &resourceLibrary);
```

Condition: (2) OS/2 Only

```
public:
IResourceId(
   unsigned long resourceId,
   const IDynamicLinkLibrary &dllLibrary);

public:
IResourceId(unsigned long resourceId);
```

You can construct objects of this class by providing the following:

- An ID and a reference to an IResourceLibrary (see page 738).
- An ID and a reference to an IDynamicLinkLibrary (see page 244).
- Only an ID. The constructor uses the resource library returned by a call to ICurrentApplication::userResourceLibrary (see page 233). This is the recommended use of the class.

Public Members

asDebugInfo

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

Overridden to provide debugging information about the class object. The current object returns asString and the results of a call to the asDebugInfo of its resource libraries.

© Copyright IBM Corp. 1994

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

Overridden to provide textual information about the class object. The current implementation returns the resource ID.

id
```
unsigned long id() const;
```

Returns the identifier used for the resource.

operator unsigned long
```
operator unsigned long() const;
```

Returns the identifier used for the resource.

resourceLibrary
```
const IResourceLibrary &resourceLibrary() const;
```

Returns a const reference to the resource library.

Figure 250. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 172. IResourceLibrary

Derivation

IBase
  IVBase
    IResourceLibrary

Header File: ireslib.hpp

The IResourceLibrary class provides the functions for loading most types of system resources.

Generally, only the User Interface Class Library itself uses this class as these components typically accept an IResourceId (see page 736) as input and do the resource loading themselves.

Constructors

```
public:
IResourceLibrary();

public:
IResourceLibrary(const IResourceLibrary &resLibrary);
```

You can only construct objects of this class using the default constructor, which does not accept any parameters, or using the copy constructor to copy one resource library to another.

Public Members

asDebugInfo

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

Provides debugging information about the class object. This function returns a string with the contents of the form: IResourceLibrary(Filename(filename)) where *filename* is the name of the resource file returned by the fileName member function.

asString

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

Provides textual information about the class object. Provides the same type of information as the member function asDebugInfo.

© Copyright IBM Corp. 1994

fileName

```
virtual IString fileName() const;
```

Returns the fully qualified file name of the resource library.

handle

```
virtual IModuleHandle handle() const;
```

Returns the handle used to load resources.

Presentation Manager Information: This is always 0 for an IResourceLibrary.

isOpen

```
virtual Boolean isOpen() const;
```

Returns the open state of the resource file. This is always true for an IResourceLibrary.

loadAccelTable

Condition: (3) AIX only

```
virtual void loadAccelTable(
               unsigned long accelTableId,
               IWindowHandle menu) const;
```

Condition: (2) OS/2 Only

```
virtual IAccelTblHandle loadAccelTable(
                         unsigned long accelTableId) const;
```

Loads an accelerator table from a resource file. The accelerator table is declared in the resource file and defines the accelerator keys.

loadBitmap

```
virtual IBitmapHandle loadBitmap(
                       unsigned long bitmapId,
                       const ISize &bitmapSize,
                       Boolean cached = true) const;

virtual IBitmapHandle loadBitmap(
                       unsigned long bitmapId,
                       Boolean cached = true) const;
```

Loads a bit map from a resource file. If you specify *cached*, the bit map is reference counted. However, if you specify a size, the bit map is loaded at that size. Reference counted refers to the concept of counting IBitmapHandles. The actual bit maps (and icons) are kept in a list with a count field that indicates how many IBitmapHandles reference the bit map. If a call is made to load a bit map and it already exists in the list, the process updates the reference count and returns the existing bit. When all references, that is all IBitmapHandles, are destructed, the bit map itself is destroyed.

Note: A match for an existing bit map does not occur unless their sizes, IDs, and resource file names are the same.

loadDialog

```
virtual IWindowHandle loadDialog(
                       unsigned long dialogId,
                       IWindow *dialogParent,
                       IWindow *dialogOwner,
                       IWinProc *dialogProcedure,
                       void *dialogCreateParameters) const;
```

Loads a dialog resource and creates a frame window.

Motif Information: AIX does not support dialogs.

loadHelpTable

```
virtual IResourceLibrary &loadHelpTable(
                           IWindow *helpInstance,
                           unsigned long helpTableId) const;
```

Loads a help table from a resource file. IHelpWindow (see page 422) provides more information on help windows.

loadIcon

```
virtual IPointerHandle loadIcon(
                        unsigned long iconId,
                        Boolean cached = true) const;
```

Loads an icon from a resource file. If you specify *cached*, the icon is reference counted. Objects of the IPointerHandle class access and manage pointer resources for a pointing device.

iconId The ID of the icon you want to load.

cached Boolean. If you set *cached* to false, you create a new copy of the icon that is never deleted. If you set *cached* to true, the library keeps a reference count for the icon. If multiple requests are made for icon, the same icon is used. If all of the references to the icon are deleted, the library also deletes the icon.

loadMenu

Condition: (3) AIX only

```
virtual void loadMenu(
              unsigned long menuId,
              IWindowHandle menuHandle,
              IMenu *parentMenu) const;
```

Condition: (2) OS/2 Only

```
virtual IWindowHandle loadMenu(
                       unsigned long menuId,
                       IWindow *menuOwner) const;
```

Loads a menu for a window from a resource file. Refer to IMenu (see page 479) for information about constructing menus.

loadMessage

```
virtual IString loadMessage(unsigned long messageId) const;
```

Loads a message resource from a resource file.

loadPointer

```
virtual IPointerHandle loadPointer(
                        unsigned long iconId,
                        Boolean cached = true) const;
```

Loads a pointer from a resource file. If you specify *cached*, the pointer is reference counted. This is the same as a call to IResourceLibrary::loadIcon (see page 740).

loadString

```
virtual IString loadString(unsigned long stringId) const;
```

Loads a string resource, using the specified string identifer from a resource file.

operator =

```
IResourceLibrary &operator =(
                    const IResourceLibrary &resLibrary);
```

Assigna one resource library to another.

sizeBitmapTo

Condition: (3) AIX only

```
static IBitmapHandle sizeBitmapTo(
                      IBitmapHandle currentHandle,
                      const ISize &newSize);
```

Scales the specified bit map to the specified *newSize*.

tryToLoadBitmap

```
virtual IBitmapHandle tryToLoadBitmap(
                       unsigned long bitmapId,
                       const ISize &bitmapSize,
                       Boolean cached = true) const;

virtual IBitmapHandle tryToLoadBitmap(
                       unsigned long bitmapId,
                       Boolean cached = true) const;
```

Loads a bit map from a resource file. If you specify *cached*, the bit map is reference counted. If you specify *bitmapSize*, the bit map is loaded using that size. If this function cannot load the bit map, it returns an IBitmapHandle with a 0 handle value.

tryToLoadIcon

```
virtual IPointerHandle tryToLoadIcon(
                        unsigned long iconId,
                        Boolean cached = true) const;
```

Loads an icon from a resource file. If you specify *cached*, the icon is reference counted. If this function cannot load the bit map, it returns an IPointerHandle with a 0 handle value.

tryToLoadMessage

```
virtual IString tryToLoadMessage(
                 unsigned long messageId) const;
```

Loads a message resource from a resource file. If this function cannot load the message, it returns an empty IString.

tryToLoadString

```
virtual IString tryToLoadString(
                 unsigned long stringId) const;
```

Loads a string resource from a resource file. If this function cannot load the string, it returns an empty IString.

Figure 251. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 173. IResourceLock

Derivation

IBase
  IVBase
    IResourceLock

Header File: ireslock.hpp

The IResourceLock class locks a resource for a specified *timeOut*. Use this class to declare a local instance of IResourceLock. If the *timeOut* is reached before the resource can be acquired, an IResourceExhausted exception is thrown.

A resource lock exists for the life of the automatic object.

Motif Information: 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.

Constructors

```
public:
IResourceLock(IResource &resource, long timeOut = -1);
```

You can only construct objects of this class by providing a reference to an IResource (see page 734) and an optional value for *timeOut*. You can specify *timeOut* in milliseconds with -1 indicating an indefinite wait and 0 requesting an immediate return.

Public Members

Figure 252. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

clearLock

```
virtual IResourceLock &clearLock();
```

Releases the resource.

setLock

```
virtual IResourceLock &setLock(long timeOut = -1);
```

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; clearLock can then be called to clear the lock and setLock called to reset it. This is not normal use of the class. Normally, you use the constructor and scope it to the instruction that require serialization.

Motif Information: The AIX timer services used to implement the resource lock *timeOut* have a resolution of one second. This function rounds the timeout value specified up to the next one second interval.

Chapter 174. IScrollBar

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          IScrollBar
```

Header File: iscroll.hpp

Objects of the IScrollBar class create and manage scroll bar control windows.

Note: The size limitation for a scroll bar is 32768.

Nested Classes

IScrollBar::Style

Constructors

```
public:
IScrollBar(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IScrollBar(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRange &scrollableItems,
   unsigned long visibleItemCount,
   unsigned long scrollBoxPosition = 1,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IScrollBar(unsigned long windowId, IWindow *parent);

public:
IScrollBar(const IWindowHandle &handle);
```

You can construct objects of this class using the following:

- 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:

© Copyright IBM Corp. 1994

Range of items	100
Visible items	25
Position	1

- 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
- The parent window
- The handle of an existing scroll bar control

windowId The scroll bar control ID.

parent The parent window.

owner The owner window.

initial (Optional) A rectangle for the scroll box control. It specifies the initial position and size of the scroll box you construct.

style (Optional) The initial style for the control. The default is classDefaultStyle (see page 756).

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.

handle The window handle of an existing scroll bar control.

Public Members

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 756) unless you have changed the style using setDefaultStyle (see page 749).

enableNotification

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

Causes the scroll bar control to send notifications to any observer objects.

foregroundColor

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

Returns the color of the shaft area.

hiliteForegroundColor

```
virtual IColor hiliteForegroundColor() const;
```

Returns the color of the scroll box area.

isHorizontal

```
Boolean isHorizontal() const;
```

If the scroll bar is horizontal, true is returned.

isVertical

```
Boolean isVertical() const;
```

If the scroll bar is vertical, true is returned.

minScrollIncrement

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

Returns the amount of the scrollable range that is scrolled by selecting the scroll buttons.

moveScrollBoxTo

```
virtual IScrollBar &moveScrollBoxTo(
                      unsigned long firstItem = 1);
```

Moves the scroll box to the specified position.

moveSizeTo

Condition: (3) AIX only

```
virtual IScrollBar &moveSizeTo(const IRectangle &rect);
```

Changes the position and size of the scroll bar.

pageScrollIncrement

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

Returns the amount of the scrollable range that is scrolled by selecting the scroll shaft.

position

Condition: (3) AIX only

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

Returns the position of the scroll bar.

prevScrollBoxPosition

Condition: (3) AIX only

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

Returns the previous position of the scroll box.

resetForegroundColor

Condition: (3) AIX only

```
virtual IScrollBar &resetForegroundColor();
```

Resets the color of the shaft area.

resetHiliteForegroundColor

Condition: (3) AIX only

```
virtual IScrollBar &resetHiliteForegroundColor();
```

Resets the color of the scroll box color area.

scrollableRange

```
virtual IRange scrollableRange() const;
```

Returns the range that the scroll bar represents.

scrollBoxPosition

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

Returns the current position of the scroll box.

scrollBoxRange

```
virtual IRange scrollBoxRange() const;
```

Returns the range in which the scroll box can be positioned. This is defined to be:

```
IRange(scrollableRange().lowerBound(),
       scrollableRange().upperBound() - visibleCount())
```

See class IRange (see page 715) for more information.

setDefaultStyle

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

Sets the default style for all subsequent scroll bars.

style Use the styles provided by IScrollBar::Styles (see page 756) to specify the default style.

setForegroundColor

Condition: (3) AIX only

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

Sets the color of the shaft area.

color Color object used to set the shaft area.

setHiliteForegroundColor

Condition: (3) AIX only

```
virtual IScrollBar &setHiliteForegroundColor(
                      const IColor &color);
```

Sets the color of the scroll box area.

color Color object used to set the specified portion of the scroll box.

setMinScrollIncrement

```
virtual IScrollBar &setMinScrollIncrement(
                      unsigned long scrollableRangeUnits = 1);
```

Sets the amount of the scrollable range that is scrolled by selecting the scroll buttons.

setPageScrollIncrement

```
virtual IScrollBar &setPageScrollIncrement(
                      unsigned long scrollableRangeUnits = 0);
```

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 scroll 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 scroll increment to 0, which prevents the page from scrolling. The User Interface Class Library avoids this possibility by setting the default page scroll increment to the minimum scroll increment in these situations.

setPrevScrollBoxPosition

Condition: (3) AIX only

```
virtual IScrollBar &setPrevScrollBoxPosition(
                      unsigned long value);
```

Sets the value of the previous scroll box position.

value Number representing the control's previous position.

Motif Information: 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, OS/2 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

```
virtual IScrollBar &setScrollableRange(
                      const IRange &minMax);
```

Sets the range of all items the scroll bar can scroll to.

setScrollBar

```
virtual IScrollBar &setScrollBar(
                      const IRange &scrollableRange,
                      unsigned long visibleCount);
```

Sets the range of all items and the number of items that are displayed. This is functionally equivalent to calling both setScrollableRange (see page 751) and setVisibleCount (see page 751).

setVisibleCount

```
virtual IScrollBar &setVisibleCount(
                      unsigned long scrollableRangeUnits);
```

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.

size

Condition: (3) AIX only

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

Returns the size of the scroll bar.

systemScrollBarWidth

```
static unsigned long systemScrollBarWidth(
                      Boolean verticalScrollBar = true);
```

Returns the system width of a vertical or horizontal scroll bar.

systemScrollBoxLength

```
static unsigned long systemScrollBoxLength(
                      Boolean verticalScrollBar = true);
```

Returns the system length of a scroll box for a vertical or horizontal scroll bar.

systemScrollButtonLength

```
static unsigned long systemScrollButtonLength(
                      Boolean verticalScrollBar = true);
```

Returns the system length of a scroll button for a vertical or horizontal scroll bar.

visibleCount

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

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.

Figure 253. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow

Figure 253. Inherited Public Members

Member function	Class
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow

Figure 253. Inherited Public Members

Member function	Class
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow

Figure 253. Inherited Public Members

Member function	Class
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the recommended minimum size of this scroll bar control.

passEventToOwner

Condition: (3) AIX only

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

Determines whether this event can be passed on to the owner of this control.

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

unregisterCallbacks

Condition: (3) AIX only

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

Removes X/Motif callbacks that were registered for the XmScrollBar widget. IWindow (see page 924) calls this function, so the User Interface Class Library application developer should not call this function directly.

Figure 254. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow

Figure 254. Inherited Protected Members

Member function	Class
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow

Figure 254. Inherited Protected Members

Member function	Class
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 175. IScrollBar::Style

Derivation

IBase
  IBitFlag
    IScrollBar::Style

Header File: iscroll.hpp

The nested class IScrollBar::Style provides a set of valid scroll bar styles for the static member functions of the class IScrollBar (see page 746). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

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.

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible | vertical.

horizontal
Causes a horizontal scroll bar to be created.

vertical
Causes a vertical scroll bar to be created.

Note: If you specify both the horizontal and vertical style, the horizontal style is ignored.

Public Members

Figure 255. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*

Figure 255. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 256. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 176. IScrollEvent

Derivation

IBase
  IVBase
    IEvent
      IScrollEvent

Header File: iscrlevt.hpp

Objects of the IScrollEvent class represent a scroll event for an IScrollBar (see page 746) object. This event is a notification for interactions with an IScrollBar object. An IScrollHandler (see page 760) object creates an IScrollEvent object and passes the event to one of its virtual functions for processing.

Constructors

```
public:
IScrollEvent(const IEvent &event);
```

Although you can construct objects of this class, typically IScrollHandler::dispatchHandlerEvent (see page 761) creates objects of this class from an object of the class IEvent (see page 275).

Public Members

newScrollBoxPosition

```
unsigned long newScrollBoxPosition() const;
```

Returns the new position of the scroll box. This is valid only for a scroll event passed to IScrollHandler::scrollBoxTrack (see page 763) or IScrollHandler::scrollBoxTrackEnd (see page 763).

scrollAmount

```
long scrollAmount() const;
```

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.

scrollBarWindow

```
IScrollBar *scrollBarWindow() const;
```


© Copyright IBM Corp. 1994

Returns a pointer to the IScrollBar (see page 746) object that the scroll event applies to.

Motif Information: This function throws an IInvalidRequest exception if the window is not a derived class of XmScrollBar (an IScrollBar).

Figure 257. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
controlHandle	IEvent
controlWindow	IEvent
dispatchingWindow	IEvent
eventId	IEvent
eventType	IEvent
handle	IEvent
operator <<	IBase*
operator <<	IVBase*
parameter1	IEvent
parameter2	IEvent
passToOwner	IEvent
result	IEvent
setControlHandle	IEvent
setDispatchingHandle	IEvent
setEventType	IEvent
setHandle	IEvent
setPassToOwner	IEvent
setResult	IEvent
window	IEvent

Note: * Application Support Class Library

Chapter 177. IScrollHandler

Derivation

IBase
  IVBase
    IHandler
      IScrollHandler

Header File: iscrlhdr.hpp

Objects of the IScrollHandler class handle events resulting from a user interacting with an IScrollBar (see page 746) 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 (see page 403) to pass the owner window to the scroll handler.

When the scroll handler receives a scroll event, it creates an IScrollEvent (see page 758) 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.

false 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 (see page 946) to process the scroll event.

Portability Considerations: The OS/2 and Motif scroll bar have slightly different characteristics. The Presentation Manager 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 version moves

© Copyright IBM Corp. 1994

the scroll box to the new location and the Motif version moves the scroll box to its current position.

Presentation Manager Information: Scroll events are not dispatched to the IScrollBar object itself, but to the IScrollBar's owner window.

Constructors

```
protected:
IScrollHandler();
```

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.

Public Members

Figure 258. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a scroll event is received, the appropriate virtual function is called.

lineDown
```
virtual Boolean lineDown(IScrollEvent &event);
```

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.

lineLeft
```
virtual Boolean lineLeft(IScrollEvent &event);
```

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.

lineRight
```
virtual Boolean lineRight(IScrollEvent &event);
```

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.

lineUp
```
virtual Boolean lineUp(IScrollEvent &event);
```

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.

moveScrollBox
```
IScrollHandler &moveScrollBox(IScrollEvent &event);
```

Positions the scroll box based on the scroll event.

pageDown
```
virtual Boolean pageDown(IScrollEvent &event);
```

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.

pageLeft
```
virtual Boolean pageLeft(IScrollEvent &event);
```

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.

pageRight
```
virtual Boolean pageRight(IScrollEvent &event);
```

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.

pageUp

```
virtual Boolean pageUp(IScrollEvent &event);
```

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.

scrollBoxTrack

```
virtual Boolean scrollBoxTrack(IScrollEvent &event);
```

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.

scrollBoxTrackEnd

```
virtual Boolean scrollBoxTrackEnd(IScrollEvent &event);
```

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.

scrollEnd

```
virtual Boolean scrollEnd(IScrollEvent &event);
```

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.

Figure 259. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 178. ISelectHandler

Derivation

IBase
  IVBase
    IHandler
      ISelectHandler

Header File: iselhdr.hpp

Objects of 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 (see page 71)
IComboBox (see page 108)
IContainerControl (see page 149)
IListBox (see page 458)
IRadioButton (see page 707)

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 (see page 403).

When the selection handler receives a selection event, it creates an IControlEvent (see page 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. Do not pass it to another handler.

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 (see page 947) to dispatch the selection event to the control's owner window.
- If this is the last handler for the owner window, call IWindow::defaultProcedure (see page 946) to process the selection event.

Motif Information: The ISelectHandler is called when ISettingButton::select and IButton::click are called for radio button and check box controls.

© Copyright IBM Corp. 1994

Presentation Manager Information: The ISelectHandler is called when ISettingButton::select is called and the object is an IRadioButton with an autoSelect style.

Constructors

```
protected:
ISelectHandler();
```

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.

Public Members

Figure 260. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a select event is received, the appropriate virtual function is called.

enter

```
virtual Boolean enter(IControlEvent &event);
```

Implemented by derived classes to process an enter event for list boxes or button objects. This function calls ISelectHandler::selected (see page 766) for button controls that can process enter events. For button controls, an enter event is only caused by a double click.

selected

```
virtual Boolean selected(IControlEvent &event);
```

Implemented by derived classes to process a selection event.

Figure 261. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 179. ISemaphoreHandle

Derivation

IBase
  IHandle
    ISemaphoreHandle

Header File: ihandle.hpp

Objects of the ISemaphoreHandle class access semaphores. An example of a semaphore is a flag in a multiple-user application that prevents simultaneous access to a file.

Motif Information: ISemaphoreHandle is an alias for a semaphore ID as used in the semop, semget, and semctl system functions.

Presentation Manager Information: ISemaphoreHandle is an alias for the OS/2 Programmer's Toolkit type HSEM.

Constructors

```
public:
ISemaphoreHandle(Value hsem = 0);
```

You can construct objects of this class from a semaphore handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the ISemaphoreHandle value.

Figure 262. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 180. ISetCanvas

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ICanvas
            ISetCanvas

Header File: isetcv.hpp

Objects of the ISetCanvas class are windows that organize their child windows in row or column decks.

Deck is the direction-independent term for a line of child windows in a set canvas window. A horizontal deck is equivalent to a row and a vertical deck is equivalent to a column. An ISetCanvas object can hold more than one deck of child windows, but the decks must be either all horizontal or all vertical.

This class arranges fixed-size child windows of similar size, such as a row of push buttons or a group of radio buttons. You can divide these child windows evenly among multiple decks by number rather than by size. ISetCanvas also sizes and positions its child windows based on their minimum sizes instead of the size of the canvas. ISetCanvas uses IWindow::minimumSize (see page 932) and IWindow::calcMinimumSize (see page 945) to size the windows.

This class provides you a number of alignment and packing option controls over how differently-sized child windows can be positioned.

Nested Classes

ISetCanvas::Style

Constructors

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

You can construct objects of this class using the following parameters:

© Copyright IBM Corp. 1994

id Window identifier of the canvas you construct.

Note: For OS/2, this value should be limited to the range 0 through 65535.

It is recommended that you:

- Give unique identifiers to all windows in the same frame window. *In the same window* means that the frame window is the first frame in their parent window chain.
- Give the client window a window ID of FID_CLIENT (0x8008).
- Do not use the other FID_xxx values.

parent The parent window of the canvas you construct. You must specify a parent window. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0.

The parent window is primarily used for visible relationships.

owner Owner window of the canvas you construct.

Note: In OS/2, the owner window is primarily used for routing notification events and unprocessed messages. OS/2 also uses the owner window chain to inherit colors.

In Motif, the owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial (Optional) The initial position and size of the canvas you construct. The default constructor is IRectangle (see page 717). The initial position is the lower-left corner of the canvas relative to the lower-left corner of the parent window.

style (Optional) Use the styles provided by IMultiCellCanvas::Style (see page 555) to specify the control's styles.

Public Members

alignment

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

Returns the enumerator for deck alignment. The returned value is an enumerator provided by Alignment (see page 776).

deckCount

```
unsigned long deckCount() const;
```

Returns the maximum number of decks used by the canvas. If your ISetCanvas has a vertical orientation, this function determines the maximum number of columns of controls the canvas has. If your set canvas has a horizontal orientation, this function determines the maximum number rows of controls the canvas has.

deckOrientation

```
DeckOrientation deckOrientation() const;
```

Returns an enumerator for deck direction. The returned value is an enumerator provided by DeckOrientation (see page 776).

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 778) unless you have changed the style using setDefaultStyle (see page 771).

groupPad

```
unsigned long groupPad() const;
```

Returns the pad width or height, which is the space before a child window in a deck with a style of IControl::group.

margin

```
ISize margin() const;
```

Returns the margin width and height, which is the space between the edge of the canvas and the outermost child windows.

moveSizeTo

Condition: (3) AIX only

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

Changes the position and size of the window.

packType

```
PackType packType() const;
```

Returns an enumerator for child window spacing in decks. The returned value is an enumerator provided by PackType (see page 777).

pad

```
ISize pad() const;
```

Returns the pad width and height, which is the space between child windows in a deck and between multiple decks.

position

Condition: (3) AIX only

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

Returns the position of the window.

setAlignment

```
virtual ISetCanvas &setAlignment(Alignment value);
```

Sets how child windows are aligned in decks.

Note: If the canvas uses a pack type of expanded, this function has no effect.

value Use the enumeration Alignment (see page 776) to specify the child window alignment.

setDeckCount

```
virtual ISetCanvas &setDeckCount(unsigned long decks);
```

Sets the maximum number of decks used by the canvas. This function has no effect if the decksByGroup style is being used.

setDeckOrientation

```
virtual ISetCanvas &setDeckOrientation(
                    DeckOrientation value);
```

Sets the direction of the decks.

value Use the enumeration DeckOrientation (see page 776) to specify the direction.

setDefaultStyle

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

Sets the default style for all subsequent canvases.

style Use the styles provided by ISetCanvas::Styles (see page 778) to specify the default style.

setGroupPad

```
virtual ISetCanvas &setGroupPad(unsigned long groupPad);
```

Sets the pad width or height to use for controls with a style of IControl::group. The default value is zero.

setLayoutDistorted

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

Treats the following as changing the layout of its child windows:

- A minimum size change for a child window
- A font change (if the set canvas has text assigned)
- An added child window
- A removed child window

setMargin

```
virtual ISetCanvas &setMargin(const ISize &margin);
```

Sets the margin width and height, which is the space between the edge of the canvas and the outermost child windows.

setPackType

```
virtual ISetCanvas &setPackType(PackType value);
```

Sets how child windows are spaced in decks.

value Use the enumeration PackType (see page 777) to specify the spacing.

setPad

```
virtual ISetCanvas &setPad(const ISize &pad);
```

Sets the pad width and height, which is the space between child windows in a deck and between multiple decks.

setText

```
virtual ISetCanvas &setText(const IResourceId &text);

virtual ISetCanvas &setText(const char *text);
```

Draws a group box with the specified text around the canvas. If necessary, the canvas is expanded to fill the width of the group box.

text The new group-box text. If you specify a null pointer or a zero-length string, an existing group box is removed.

size

Condition: (3) AIX only

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

Returns the size of the window.

text

```
virtual IString text() const;
```

Returns the text of the group box. If the canvas has no surrounding group box, an empty IString is returned.

Figure 263. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	ICanvas
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
defaultPushButton	ICanvas
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow

Figure 263. Inherited Public Members

Member function	Class
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isTabStop	ICanvas
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
matchForMnemonic	ICanvas
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier

Figure 263. Inherited Public Members

Member function	Class
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
origDefaultButtonHandle	ICanvas
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow

Figure 263. Inherited Public Members

Member function	Class
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ICanvas
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

expandForText

```
virtual ISetCanvas &expandForText();
```

Determines how much larger the canvas must be to include a group box with text.

layout

```
virtual ISetCanvas &layout();
```

Positions and sizes child windows.

Figure 264. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
areChildrenReversed	ICanvas
calcMinimumSize	IWindow
calcMinimumSize	ICanvas
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
fixupChildren	ICanvas
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
layout	ICanvas
layoutSize	ICanvas
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ICanvas
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	ICanvas
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier

Figure 264. Inherited Protected Members

Member function	Class
removeObserver	IWindow
resetColor	IWindow
setLayoutSize	ICanvas
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	ICanvas
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

```
public:
enum Alignment
  {
  topLeft,
  topCenter,
  topRight,
  centerLeft,
  centerCenter,
  centerRight,
  bottomLeft,
  bottomCenter,
  bottomRight,
  };
```

Use the following enumerators to specify the alignment of child windows in a deck:

topLeft
Aligns the child windows in the canvas to the top and left edge in the deck.

topCenter
Aligns the child windows in the canvas to the top edge, centered horizontally in the deck.

topRight
Aligns the child windows in the canvas to the top and right edge in the deck.

centerLeft
Aligns the child windows in the canvas to the left edge, centered vertically in the deck.

centerCenter
Aligns the child windows in the canvas to the center of the deck.

centerRight
Aligns the child windows in the canvas to the right edge, centered vertically in the deck.

bottomLeft
Aligns the child windows in the canvas to the bottom and left edge in the deck.

bottomCenter
Aligns the child windows in the canvas to the bottom edge, centered vertically in the deck.

bottomRight
Aligns the child windows in the canvas to the bottom and right edge in the deck.

DeckOrientation

```
public:
enum DeckOrientation
  {
  horizontal,
  vertical,
  };
```

Use the following enumerators to specify the direction in which to orient the deck of child windows:

horizontal
Arranges the child windows in the canvas horizontally in rows, from left to right. Decks of rows are arranged from top to bottom.

vertical
Arranges the child windows in the canvas vertically in columns, from top to bottom. Decks of columns are arranged from left to right.

PackType

```
public:
enum PackType
  {
  tight,
  even,
  expanded,
  };
```

Use these enumerators to specify the spacing between child windows in decks:

tight
Sizes the child windows to their minimum size and separates them within a deck by the pad amount only.

even
Sizes child windows to their minimum size and spaces them so that corresponding windows in each deck are aligned. This gives the appearance of their being arranged in both rows and columns. As a result, child windows within a deck can be separated by more than the pad amount. This option has no affect if the set canvas has only one deck.

expanded
Similar to the enumerator even, except the child windows are expanded, rather than spaced by more than the pad amount. Child windows are expanded to the same size as the largest child window.

Chapter 181. ISetCanvas::Style

Derivation

IBase
  IBitFlag
    ISetCanvas::Style

Header File: isetcv.hpp

The nested class ISetCanvas::Style provides a set of valid canvas styles for the defaultStyle and setDefaultStyle functions, and for the constructor of the ISetCanvas (see page 768) class. You can use these styles with the styles defined by the following nested class:

ICanvas::Style (see page 70)
IWindow::Style (see page 957)

Note: To avoid conflicts that are noted in some of the alignment, deck, and pack styles, you can implement the wanted characteristics for the canvas by using the setAlignment, setDeckOrientation, and setPackType member functions.

Styles

bottomAlign
Aligns child windows at the bottom edge of a horizontal deck or the bottom edge of a row between evenly packed vertical decks.

Note: If you use this style with both packTight *and* verticalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with topAlign or centerVerticalAlign.

centerAlign
Centers child windows horizontally within a vertical deck or within a column between evenly packed horizontal decks.

Note: If you use this style with both packTight *and* horizontalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with leftAlign or rightAlign.

centerVerticalAlign
Centers child windows vertically within a horizontal deck or within a row between evenly packed vertical decks.

Note: If you use this style with both packTight *and* verticalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with topAlign or bottomAlign.

© Copyright IBM Corp. 1994

classDefaultStyle
Specifies the original default style for this class, which is horizontalDecks | packTight | leftAlign | topAlign | IWindow::visible.

horizontalDecks
Organizes child windows in rows. Multiple rows are arranged from top to bottom.

Note: You cannot specify this style with verticalDecks.

leftAlign
Aligns child windows at the left edge of a vertical deck or the left edge of a column between evenly packed horizontal decks.

Note: If you use this style with both packTight *and* horizontalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with rightAlign or centerAlign.

packEven
Sizes child windows to their minimum size and spaces them such that corresponding windows in each deck are aligned. This gives the child windows the appearance of being arranged in both rows and columns. As a result, child windows within a deck can be separated by more than the pad amount.

Note: If the set canvas has only one deck, this style has no effect. You cannot specify this style with packTight or packExpanded.

packExpanded
Sizes child windows in a way similar to that of packEven, except the child windows are enlarged rather than spaced by more than the pad amount. All smaller child windows are enlarged to the same size as the largest child window.

Note: You cannot specify this style with packTight or packEven.

packTight
Sizes child windows to their minimum size and separates them within a deck by the pad amount only.

Note: You cannot specify this style with packEven or packExpanded.

rightAlign
Aligns child windows at the right edge of a vertical deck or the right edge of a column between evenly packed horizontal decks.

Note: If you use this style with both packTight *and* horizontalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with leftAlign or centerAlign.

topAlign
Aligns child windows at the top edge of a horizontal deck or a row between evenly packed vertical decks.

Note: If you use this style with both packTight *and* verticalDecks, it is ignored. If you use this style with packExpanded, it is ignored. You cannot specify this style with bottomAlign or centerVerticalAlign.

verticalDecks

Organizes child windows in columns. Multiple columns are arranged from left to right.

Note: You cannot specify this style with horizontalDecks.

Public Members

Figure 265. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 266. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 182. ISettingButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              ISettingButton

Header File: isetbut.hpp

The ISettingButton class is an abstract base class for the following control window classes:

- ICheckBox (see page 71)
- IRadioButton (see page 707)

Constructors

```
public:
ISettingButton();
```

Because this class is an abstract base class, you cannot construct objects of this class.

Public Members

deselect

```
virtual ISettingButton &deselect();
```

Deselects the button.

disableAutoSelect

```
virtual ISettingButton &disableAutoSelect() = 0;
```

Disables the automatic selection style for a button.

Motif Information: AIX does not support this function, therefore it has no effect.

enableAutoSelect

```
virtual ISettingButton &enableAutoSelect(
                            Boolean enable) = 0;
```

Enables the automatic selection style for a button.

© Copyright IBM Corp. 1994

Motif Information: AIX does not support this function, therefore it has no effect.

enableNotification

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

Causes the button to send notifications to any observer objects.

isAutoSelect

```
virtual Boolean isAutoSelect() const = 0;
```

Returns true if the automatic selection style for a button is set on.

Motif Information: The AIX version is always autoSelect; therefore this function returns true.

isSelected

```
Boolean isSelected() const;
```

If the button is selected, true is returned.

select

```
virtual ISettingButton &select(Boolean select = true);
```

Selects the button. This function does not call ISelectHandler. Use IButton::click instead to ensure that ISelectHandler is called.

Figure 267. Inherited Public Members

Member function	Class
activeColor	IWindow
allowsMouseClickFocus	IButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IButton
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
click	IButton

Figure 267. Inherited Public Members

Member function	Class
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disabledForegroundColor	IButton
disableGroup	IControl
disableMouseClickFocus	IButton
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl

Figure 267. Inherited Public Members

Member function	Class
enableMouseClickFocus	IButton
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IButton
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
highlight	IButton
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IButton
hiliteForegroundColor	IWindow
hiliteForegroundColor	IButton
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isHighlighted	IButton
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow

Figure 267. Inherited Public Members

Member function	Class
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow

Figure 267. Inherited Public Members

Member function	Class
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
setText	IButton
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
unhighlight	IButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

passEventToOwner

Condition: (3) AIX only

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

Determines if the event is passed on to the owner.

Figure 268. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow

Figure 268. Inherited Protected Members

Member function	Class
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow

Figure 268. Inherited Protected Members

Member function	Class
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 183. ISharedResource

Derivation

IBase
  IVBase
    IResource
      ISharedResource

Header File: ireslock.hpp

Use the ISharedResource class to define a resource that can be shared across multiple processes using a name you provide on construction.

Portability Considerations: You should use a keyName value that is a valid file name in all of the target environments to ensure portability of your code.

Motif Information: You might use ISharedResource objects to serialize resource access with applications which 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 class library. To help you understand how the semaphore object is used by the class library, the basic interaction with the system 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 indicating the number of ISharedResource objects which are referencing this semaphore set. The constructor increments this counter and the destructor decrements it. When the counter reaches 0 the destructor calls the semctl subroutine to delete the semaphore set from the system.

The constructor determines the semaphoreID 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 a existing file, the parameter is used as a file name for the ftok call and is the basis of the semaphoreID 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 the keyName function 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.

© Copyright IBM Corp. 1994

Constructors

```
public:
ISharedResource(const char *keyName);
```

You can only construct objects of this class by providing a character string that identifies the resource to be shared.

Public Members

keyName

```
IString keyName();
```

Returns the name used to identify this public resource.

Figure 269. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
lock	IResource
operator <<	IBase*
operator <<	IVBase*
unlock	IResource

Note: * Application Support Class Library

Protected Members

handle

```
ISemaphoreHandle &handle();
```

Returns the handle for the resource.

Figure 270. Inherited Protected Members

Member function	Class
handle	IResource

Note: * Application Support Class Library

Chapter 184. IShowListHandler

Derivation

IBase
  IVBase
    IHandler
      IShowListHandler

Header File: islhdr.hpp

Objects of IShowListHandler derived classes process IControlEvent events for objects of the class IComboBox (see page 108). 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 (see page 403) 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 (see page 227) object and routes that object to the IShowListHandler::listShown (see page 789) virtual function. You should 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 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.
- If this is the last handler for the combination box, call IWindow::dispatch (see page 947) 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 (see page 946) to process the show list event.

Constructors

```
protected:
IShowListHandler();
```


© Copyright IBM Corp. 1994

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.

Public Members

Figure 271. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

dispatchHandlerEvent

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

If a show list event is received, the appropriate virtual function is called.

listShown

```
virtual Boolean listShown(IControlEvent &event) = 0;
```

Implemented by derived classes to process show list events.

Figure 272. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 185. ISize

Derivation

IBase
  IPair
    ISize

Header File: ipoint.hpp

Objects of the ISize class use their coordinates to represent a rectangular size, in horizontal and vertical dimensions.

Presentation Manager Information: You can also construct objects of this class using:

- A Presentation Manager Toolkit SIZEL structure.
- A Presentation Manager Toolkit RECTL structure; in this case, the resulting ISize object represents the size of the RECTL.

Constructors

```
public:
ISize();

public:
ISize(const IPair &pair);

public:
ISize(Coord width, Coord height);

public:
ISize(const struct _SIZEL &sizl);

public:
ISize(const struct _RECTL &rcl);
```

You can construct objects of this class from:

- The default constructor. The resulting size has a width and height of 0.
- Another ordered pair of coordinates.
- Any specified width and height.

Public Members

asSIZEL

```
struct _SIZEL asSIZEL() const;
```

Returns the ISize as a Presentation Manager Toolkit SIZEL structure.

© Copyright IBM Corp. 1994

height
```
Coord height() const;
```

Returns the height represented by the ISize object.

setHeight
```
ISize &setHeight(Coord cy);
```

Sets the size's height.

setWidth
```
ISize &setWidth(Coord cx);
```

Sets the size's width.

width
```
Coord width() const;
```

Returns the width represented by the ISize object.

Figure 273. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IPair
asString	IBase*
asString	IPair
coord1	IPair
coord2	IPair
distanceFrom	IPair
dotProduct	IPair
maximum	IPair
minimum	IPair
operator !=	IPair
operator %	IPair
operator %=	IPair
operator *	IPair
operator *=	IPair
operator +	IPair
operator +=	IPair
operator -	IPair
operator -=	IPair
operator /	IPair
operator /=	IPair
operator <	IPair
operator <<	IBase*
operator <<	IPair

Figure 273. Inherited Public Members

Member function	Class
operator <=	IPair
operator ==	IPair
operator >	IPair
operator >=	IPair
scaleBy	IPair
scaledBy	IPair
setCoord1	IPair
setCoord2	IPair
transpose	IPair

Note: * Application Support Class Library

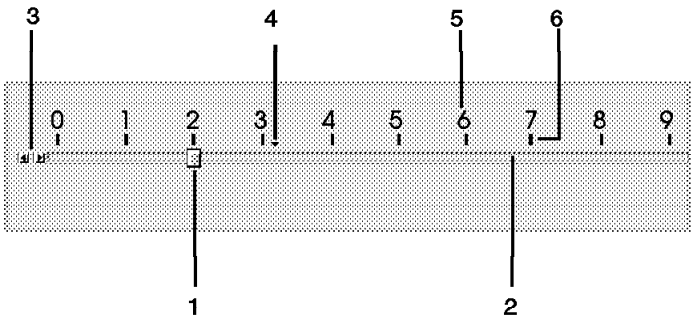
Chapter 186. ISlider

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          IProgressIndicator
            ISlider

Header File: islider.hpp

Objects of the ISlider class let a user 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.

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.

© Copyright IBM Corp. 1994

Because the slider is interactive, detents and buttons are provided and the slider arm is larger so it might be used.

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 (see page 247)
- IFocusHandler (see page 307)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)
- IKeyboardHandler (see page 455)
- IMouseClickHandler (see page 539)

Motif Information: The AIX version does not support detents.

The ISlider constructor creates objects of this class using several 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.
- XmSeparator widgets are created for each widget during the construction of the ISlider 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.

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 snapToTick style is implemented by attaching an XmNvalueChanged callback to the scale widget.

Nested Classes

ISlider::Style

Constructors

```
public:
ISlider(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initial,
   unsigned long numberOfTicks,
   unsigned long tickSpacing = 0,
   const Style &style = defaultStyle ( ));

public:
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 ( ));
```

Condition: (2) OS/2 Only

```
public:
ISlider(unsigned long windowId, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
ISlider(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Using the 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
(Optional) The number of pixels between ticks. The default is 0. Using 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.

style (Optional) The style of the control.

- Specifying the number of ticks and spacing for both scale 1 and scale 2

  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 default is 0. The number of pixels between ticks on scale 2.

- Specifying a parent window and a slider control ID.

- Specifying a window handle

  handle The window handle of the slider control.

Public Members

addDetent

Condition: (1) Ignored on AIX

```
virtual unsigned long addDetent(unsigned long offset);
```

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.

armSize

```
ISize armSize() const;
```

Returns the size of the arm in pixels.

buttonsPosition

```
ButtonsPosition buttonsPosition() const;
```

Returns the current position of the slider's buttons for this slider object. The returned value is an enumerator provided by ButtonsPosition (see page 800).

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 801) unless you have changed the style using setDefaultStyle (see page 796).

detentPosition

Condition: (1) Ignored on AIX

```
virtual unsigned long detentPosition(
                      unsigned long detentId) const;
```

Returns the offset of a detent from the home position, in pixels.

detendId You must specify the detent's ID.

removeDetent

Condition: (1) Ignored on AIX

```
virtual ISlider &removeDetent(unsigned long detentId);
```

Removes the specified detent from the slider.

detentId Detent to remove.

setArmSize

Condition: (1) Ignored on AIX

```
virtual ISlider &setArmSize(const ISize &sliderArm);
```

Sets the size of the arm in pixels.

sliderArm Object that represents the horizontal and vertical dimensions.

setDefaultStyle

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

Sets the default style for all subsequent sliders.

style Use the styles provided by ISlider::Styles (see page 801) to specify the default style.

Figure 274. Inherited Public Members

Member function	Class
activeColor	IWindow
alignment	IProgressIndicator
armPixelOffset	IProgressIndicator
armRange	IProgressIndicator
armTickOffset	IProgressIndicator
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	IProgressIndicator
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableDrawItem	IProgressIndicator
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableRibbonStrip	IProgressIndicator
disableSnapToTick	IProgressIndicator
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableDrawItem	IProgressIndicator
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableRibbonStrip	IProgressIndicator
enableSnapToTick	IProgressIndicator
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IProgressIndicator

Figure 274. Inherited Public Members

Member function	Class
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
homePosition	IProgressIndicator
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isDrawItemEnabled	IProgressIndicator
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isRibbonStripEnabled	IProgressIndicator
isShowing	IWindow
isSnapToTickEnabled	IProgressIndicator
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVertical	IProgressIndicator
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveArmToPixel	IProgressIndicator
moveArmToTick	IProgressIndicator
moveSizeTo	IWindow
moveSizeTo	IProgressIndicator
moveTo	IWindow
notifyObservers	INotifier

Figure 274. Inherited Public Members

Member function	Class
notifyObservers	IWindow
numberOfTicks	IProgressIndicator
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
primaryScale	IProgressIndicator
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBackgroundColor	IProgressIndicator
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setForegroundColor	IProgressIndicator

Figure 274. Inherited Public Members

Member function	Class
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setHomePosition	IProgressIndicator
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setPrimaryScale	IProgressIndicator
setShadowColor	IWindow
setShaftBreadth	IProgressIndicator
setShaftPosition	IProgressIndicator
setTickLength	IProgressIndicator
setTicks	IProgressIndicator
setTickText	IProgressIndicator
shadowColor	IWindow
shaftPosition	IProgressIndicator
shaftSize	IProgressIndicator
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
tickLength	IProgressIndicator
tickPosition	IProgressIndicator
tickSpacing	IProgressIndicator
tickText	IProgressIndicator
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 275. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
calcMinimumSize	IProgressIndicator
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
initialize	IProgressIndicator
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IProgressIndicator
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IProgressIndicator
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

ButtonsPosition

```
public:
enum ButtonsPosition
  {
  top,
  bottom,
  left,
  right,
  none,
  };
```

Use these enumerators to specify the placement of the slider's buttons. Use the buttons to move the slider arm one tick 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.

none
The slider has no buttons.

Chapter 187. ISlider::Style

Derivation

IBase
  IBitFlag
    ISlider::Style

Header File: islider.hpp

The nested class ISlider::Style provides a set of valid slider styles for the static member functions of the class ISlider (see page 792). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IProgressIndicator::Style (see page 695)

Styles

buttonsBottom
Places the slider's buttons below the slider shaft.

buttonsLeft
Places the slider's buttons to the left of the slider shaft.

buttonsRight
Places the slider's buttons to the right of the slider shaft.

buttonsTop
Places the slider's buttons above the slider shaft.

classDefaultStyle
Specifies the original default style for this class, which is IWindow::visible | IProgressIndicator::horizontal | IProgressIndicator::alignCentered | IProgressIndicator::homeLeft | IProgressIndicator::primaryScale1 | buttonsLeft.

Public Members

Figure 276. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag

Figure 276. Inherited Public Members

Member function	Class
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 277. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 188. ISpinButton

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ISpinButton

Header File: ispinbt.hpp

Objects of the ISpinButton class create and manage spin button control windows.

Note: ISpinButton and its nested classes ISpinButton::Cursor and ISpinButton::Style are obsolete. The User Interface Class Library maintains these classes only for backward compatibility. Instead, use the following classes (and their nested classes):

- IBaseSpinButton (see page 32)
- INumericSpinButton (see page 612)
- ITextSpinButton (see page 878)

Motif Information: The AIX release supports only the first constructor, which creates an object of this class from a control ID, a parent window, an owner window, a rectangle object, and a style object.

The ISpinButton constructor creates objects of this class using several Motif widgets:

- An XmForm widget is created with a single line XmText child.
- If the spin button has the style ISpinButton::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 ISpinButton object via private callbacks and a default handler. The ISpinButton class uses a default handler derived from the class IKeyboardHandler. Therefore, you should attach user-defined handlers derived from IKeyboardHandler to the ISpinButton object rather than to its owner window. This enables events to be dispatched to user-defined handlers prior to the default handler processing.

Nested Classes

ISpinButton::Cursor
ISpinButton::Style

© Copyright IBM Corp. 1994

Constructors

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

Condition: (2) OS/2 Only

```
public:
ISpinButton(unsigned long id, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
ISpinButton(const IWindowHandle &handle);
```

You can construct objects of this class using the following ways:

- The parent window, owner window, optional size and location, and optional style parameters.
- The parent window.
- The handle of an existing spin button.

windowId A unique ID for the spin button control.

parent The parent window.

owner The owner window.

initial A rectangle defining the size and placement of the spin button window.

style (Optional) The style of the control.

id A unique ID for the spin button control.

handle The window handle of an existing spin button control.

Public Members

add

```
virtual ISpinButton &add(
                      const IResourceId &item,
                      Cursor &cursor,
                      Boolean immediateUpdate = false);

virtual ISpinButton &add(
                      const char *string,
                      Cursor &cursor,
                      Boolean immediateUpdate = false);
```

Adds a new item at the cursor location and sets the cursor on it.

If you use this function on a numeric spin button, an exception is generated.

string A text string to add to the alphanumeric spin button.

cursor An ISpinButton::Cursor (see page 817) object specifying the position to insert the string. The cursor will update to point to the added string.

item The resource ID of the item to add to the alphanumeric spin button.

immediateUpdate
The default is false.

addAsFirst

```
virtual ISpinButton &addAsFirst(
                        const IResourceId &item,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);

virtual ISpinButton &addAsFirst(
                        const char *string,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);
```

Adds the item as the first item and sets the cursor on it. You can use this function to add an item to an empty text ISpinButton object. The only other way to add an item to an empty text ISpinButton object is to use ISpinButton::addAsLast (see page 805).

If you use this function on a numeric spin button, an exception is generated.

string A text string to add to the alphanumeric spin button.

cursor An ISpinButton::Cursor (see page 817) object specifying the position to insert the string. The cursor will update to point to the added string.

item The resource ID of the item to be added to the alphanumeric spin button.

immediateUpdate
The default is false.

addAsLast

```
virtual ISpinButton &addAsLast(
                        const char *string,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);

virtual ISpinButton &addAsLast(
                        const IResourceId &item,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);
```

Adds the item as the last item and sets the cursor on it. You can use this function to add an item to an empty text ISpinButton object. The only other way to add an item to an empty text ISpinButton object is to use ISpinButton::addAsFirst (see page 805).

If you use this function on a numeric spin button, an exception is generated.

string A text string to add to the alphanumeric spin button.

cursor An ISpinButton::Cursor (see page 817) object specifying the position to insert the string. The cursor will update to point to the added string.

item The resource ID of the item to add to the alphanumeric spin button.

immediateUpdate
The default is false.

addAsNext

```
virtual ISpinButton &addAsNext(
                      const IResourceId &item,
                      Cursor &cursor,
                      Boolean immediateUpdate = false);

virtual ISpinButton &addAsNext(
                      const char *string,
                      Cursor &cursor,
                      Boolean immediateUpdate = false);
```

Adds the item as the next item and sets the cursor on it.

If you use this function on a numeric spin button, an exception is generated.

string A text string to add to the alphanumeric spin button.

cursor An ISpinButton::Cursor (see page 817) object specifying the position to insert the string. The cursor will update to point to the added string.

item The resource ID of the item to add to the alphanumeric spin button.

immediateUpdate
The default is false.

alignment

Condition: (1) Ignored on AIX

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

Returns the current text alignment in the spin field of this spin button object. The returned value is an enumerator provided by Alignment (see page 816).

Motif Information: This function always returns left.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 819) unless you have changed the style using setDefaultStyle (see page 810).

disableBorder

```
virtual ISpinButton &disableBorder();
```

Suppresses the drawing of a border.

elementAt

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

Returns the string at the cursor position.

If you use this function on a numeric spin button, an exception is generated.

cursor An ISpinButton::Cursor object specifying the position of the string.

enable

Condition: (2) OS/2 Only

```
virtual ISpinButton &enable(Boolean enableControl = true);
```

Enables or disables the window to accept keyboard and mouse input.

enableBorder

```
virtual ISpinButton &enableBorder(Boolean enable = true);
```

Draws a border.

foregroundColor

Condition: (2) OS/2 Only

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

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

inputTypeAllowed

```
Type inputTypeAllowed() const;
```

Returns the data type permitted in the spin field. The returned value is an enumerator provided by Type (see page 816).

isBorder

```
Boolean isBorder() const;
```

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

isFastSpin

```
Boolean isFastSpin() const;
```

If the spin speed is doubled every two seconds, true is returned.

isMaster

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

If the spin button is a master, true is returned.

isReadOnly

```
Boolean isReadOnly() const;
```

If the user is not permitted to type in the spin field, true is returned.

isServant

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

If the spin button is a servant, true is returned.

isValid

```
virtual Boolean isValid(Boolean updateContents = false);
```

If the contents of the spin field are within the number range or match one of the text values, true is returned.

updateContents

If you set updateContent to true and the spin button contains the following, the spin field is updated:

- A number range and the spin field value is below the range, the spin field is set to the lower bound.
- A number range and the spin field value is above the range, the spin field is set to the upper bound.
- Text values and the spin field contents match a spin text value but the case is different, the spin field is set to the actual spin text value.
- Text values and the spin field contents do not match a spin text value, the spin field is not updated.

range

```
virtual IRange range() const;
```

Returns the lower and upper range of the spin button.

If you use this function on an alphanumeric spin button, an exception is generated.

refreshText

```
virtual ISpinButton &refreshText();
```

Updates the visible items in an alphanumeric spin button with the buffered text items. This function provides the ability to add, remove, and replace multiple text entries before applying the changes to the spin button. We recommend that you do this when you add a large quantity of text. If you set *immediateUpdate* to true in the add, remove, and replace functions, the spin button is updated immediately.

If you use this function on a numeric spin button, an exception is generated.

removeAll

```
virtual ISpinButton &removeAll();
```

Removes all text in the spin button.

If you use this function on a numeric spin button, an exception is generated.

removeAt

```
virtual ISpinButton &removeAt(
                        Cursor &cursor,
                        Boolean immediateUpdate = false);
```

Removes the item at the specified cursor's position and sets the cursor to the item preceding the removed item.

If you use this function on a numeric spin button, an exception is generated.

replaceAt

```
virtual ISpinButton &replaceAt(
                        const char *newString,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);

virtual ISpinButton &replaceAt(
                        const IResourceId &item,
                        Cursor &cursor,
                        Boolean immediateUpdate = false);
```

Replaces the string at the cursor position.

If you use this function on a numeric spin button, an exception is generated.

setAlignment

Condition: (1) Ignored on AIX

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

Changes the text alignment in the spin field.

alignment The default is left. Use the enumeration Alignment (see page 816) to specify the alignment of the text.

setCurrent

```
virtual ISpinButton &setCurrent(long aValue);

virtual ISpinButton &setCurrent(const Cursor &cursor);
```

For spin buttons containing text values, sets the spin field to the item at the specified cursor position and calls ISpinButton::refreshText (see page 809) to update the spin button.

For spin buttons containing a number range, sets the spin field to the specified value.

This function is the same as ISpinButton::spinTo (see page 812).

setDefaultStyle

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

Sets the default style for all subsequent spin buttons.

style Use the styles provided by ISpinButton::Styles (see page 819) to specify the default style.

setFastSpin

```
virtual ISpinButton &setFastSpin(Boolean fast = true);
```

Causes the spin button to increase the spin speed. The speed doubles every two seconds.

Presentation Manager Information: Do not call this member function from a master spin button that has servants spun from it.

setInputType

```
virtual ISpinButton &setInputType(
                      Type inputType = alphanumeric);
```

Changes the permitted input type for the spin field.

inputType The default is alphanumeric. Use the enumeration Type (see page 816) to specify the input type for the spin field.

setLimit

```
virtual ISpinButton &setLimit(unsigned long aNumber);
```

Sets the number of characters permitted in the spin field. The original value at construction is 255.

Note: If you call this function with a limit that does not allow all numbers in the spin button range to display, unpredictable results might occur when the button is spun.

If you use this function on a numeric spin button, an exception is generated.

setMaster

```
virtual ISpinButton &setMaster(ISpinButton &master);
```

Defines a servant spin button's master. The spin button upon which you call this function *must* be a servant.

setRange

```
virtual ISpinButton &setRange(
                      const IRange &aNewRange,
                      Boolean override = false);
```

Sets the lower and upper range of the spin button numbers.

If you use this function on an alphanumeric spin button, an exception is generated.

aNewRange
: An inclusive range of IPair values between specified lower and upper boundaries.

override
: The default is false. 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.

setReadOnly

```
virtual ISpinButton &setReadOnly(Boolean readOnly = true);
```

Disables or enables input in the spin field.

setText

```
virtual ISpinButton &setText(const IResourceId &item);

virtual ISpinButton &setText(const char *string);
```

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.

If you use this function on a numeric spin button, an exception is generated.

setValue

```
virtual ISpinButton &setValue(long aValue);
```

Sets the displayed value of the spin field, regardless of the validity of the number.

If you use this function on an alphanumeric spin button, an exception is generated.

spinDown

```
virtual ISpinButton &spinDown(unsigned long spinBy = 1);
```

Spins the button down the specified number of times.

spinTo

```
virtual ISpinButton &spinTo(long aValue);

virtual ISpinButton &spinTo(const Cursor &cursor);
```

For spin buttons containing text values, sets the spin field to the item at the specified cursor position and calls ISpinButton::refreshText (see page 809) to update the spin button.

cursor An ISpinButton::Cursor (see page 817) object specifying the position to insert the string. The cursor is updated to point to the added string.

For spin buttons containing a number range, sets the spin field to the specified value.

aValue A number in the range.

This function is the same as ISpinButton::setCurrent (see page 810).

spinUp

```
virtual ISpinButton &spinUp(unsigned long spinBy = 1);
```

Spins the button up the specified number of times.

text

```
virtual IString text() const;
```

Returns the displayed contents of the spin field.

If you use this function on a numeric spin button, an exception is generated.

topHandle

Condition: (3) AIX only

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

Returns the top-most manager widget of the spin button. The IWindowHandle (see page 960) returned is used for sizing, positioning, and managing the control.

value

```
virtual long value() const;
```

Returns the current value displayed in the spin field.

If you use this function on an alphanumeric spin button, an exception is generated.

Figure 278. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow

Figure 278. Inherited Public Members

Member function	Class
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier

Figure 278. Inherited Public Members

Member function	Class
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow

Figure 278. Inherited Public Members

Member function	Class
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the recommended minimum size of this spin button control. The size is based on the font and text limit.

registerCallbacks

Condition: (3) AIX only

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

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.

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

unregisterCallbacks

Condition: (3) AIX only

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

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

Figure 279. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow

Figure 279. Inherited Protected Members

Member function	Class
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow

Figure 279. Inherited Protected Members

Member function	Class
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

Condition: (1) Ignored on AIX

```
public:
enum Alignment
  {
  left,
  right,
  center,
  };
```

Use these enumerators to specify the alignment of text in the spin field:

left
Left-justifies the text in the spin field.

center
Centers the text in the spin field.

right
Right-justifies the text in the spin field.

Motif Information: The AIX release ignores the Alignment enumeration. All spin buttons have left alignment.

Type

```
public:
enum Type
  {
  alphanumeric,
  numeric,
  };
```

Use these enumerators to specify the types of values users can enter in the spin button field:

alphanumeric
Specifies that the spin button field contains a list of text.

numeric
Specifies that the spin button field contains a range of numbers.

Chapter 189. ISpinButton::Cursor

Derivation

IBase
  IVBase
    ISpinButton::Cursor

Header File: ispinbt.hpp

Objects of the ISpinButton::Cursor class create and manage the cursor for an ISpinButton (see page 803) object that has text. 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 spin button, one item at a time. It is not necessary to use a cursor in a spin button that has a number range.

Constructors

```
public:
Cursor(const ISpinButton &spinButton);
```

You can only construct objects of this class from an object of ISpinButton (see page 803).

Public Members

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

Flags this cursor as not valid.

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

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

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

Points to the first item in the text list.

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

Points to the last item in the text list.

© Copyright IBM Corp. 1994

setToNext

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

Points to the next item in the text list. If there is none, the cursor is invalidated.

setToPrevious

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

Points to the previous item in the text list. If there is none, the cursor is invalidated.

Figure 280. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 190. ISpinButton::Style

Derivation

IBase
  IBitFlag
    ISpinButton::Style

Header File: ispinbt.hpp

The nested class ISpinButton::Style provides a set of valid spin button styles for the static member functions of the class ISpinButton (see page 803). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

allCharacters
Permits any character to be typed in the spin field.

centerAlign
Centers the text.

classDefaultStyle
Specifies the original default style for this class, which is master | leftAlign | allCharacter | IWindow::visible.

fastSpin
Enables the spin button to increase the spin speed with time. The speed doubles every two seconds.

leftAlign
Left-justifies the text.

master
Specifies that the spin button component consist of the following:

- At least one single line entry field (SLE), or spin field
- Two arrows: the Up arrow and the Down arrow

When a 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.

noBorder
Suppresses the drawing of a border.

© Copyright IBM Corp. 1994

numericOnly
Specifies that only digits from 0 to 9 and the minus sign (-) can be typed in the spin field.

padWithZeros
Pads the displayed number with zeros at the front.

readOnly
Prevents entering input into the spin field.

rightAlign
Right-justifies the text.

servant
Allows you to create a multiple-field spin button by spinning servants from the master. To use this style, you must call ISpinButton::setMaster (see page 811) to define the master spin field.

Motif Information: AIX does not support the centerAlign and rightAlign styles. Spin button text is always left aligned.

Public Members

Figure 281. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 282. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 191. ISpinHandler

Derivation

IBase
  IVBase
    IHandler
      ISpinHandler

Header File: ispinhdr.hpp

Objects of the ISpinHandler class process spin events for objects of the following classes:

- ISpinButton (see page 803)
- INumericSpinButton (see page 612)
- ITextSpinButton (see page 878)

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 (see page 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 (see page 403) 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 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.
- If this is the last handler for the spin button, call IWindow::dispatch (see page 947) 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 (see page 946) to process the spin event.

  Note: There is no defaultProcedure in the AIX environment.

© Copyright IBM Corp. 1994

Constructors

```
protected:
ISpinHandler();
```

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.

Public Members

Figure 283. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IHandler
asString	IBase*
asString	IVBase*
asString	IHandler
disable	IHandler
enable	IHandler
handleEventsFor	IHandler
isEnabled	IHandler
operator <<	IBase*
operator <<	IVBase*
stopHandlingEventsFor	IHandler

Note: * Application Support Class Library

Protected Members

arrowDown

```
virtual Boolean arrowDown(IControlEvent &event);
```

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.

arrowUp

```
virtual Boolean arrowUp(IControlEvent &event);
```

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.

dispatchHandlerEvent

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

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.

spinEnded

```
virtual Boolean spinEnded(IControlEvent &event);
```

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.

Figure 284. Inherited Protected Members

Member function	Class
defaultProcedure	IHandler
dispatchHandlerEvent	IHandler

Note: * Application Support Class Library

Chapter 192. ISplitCanvas

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ICanvas
            ISplitCanvas

Header File: isplitcv.hpp

Objects of the ISplitCanvas class provide a way to split a window into two or more window panes by creating and adding controls. The number of controls you add determines the number of panes in the split canvas.

You can separate the window panes using split bars (the default style). By dragging the split bars, a user can dynamically change the visible amount of each pane.

You can stack the controls vertically (the default) or horizontally.

Typically, you create a split canvas control with a frame window or another split canvas window as its parent and owner. Any window can serve this purpose. In addition, you might want to call IFrameWindow::setClient (see page 363) to make the split canvas control the client window of a frame window. Then you can add other controls, such as list boxes, push buttons, or even another split canvas by creating each control with a split canvas like its parent.

Note: You can make an IFrameWindow the child window of a split canvas *only* if it is the only child window of the canvas. You can, however, make the IFrameWindow the only child window of an IMultiCellCanvas by placing the IFrameWindow in a single expandable row and column and making the multiple-cell canvas one of many child windows of the split canvas.

Do not change the z-order of the child windows of a split canvas without refreshing the canvas. Doing so results in unpredictable behavior.

Controls are positioned in the window panes of a split canvas in the order you create them in, from left to right in a vertically split canvas and from top to bottom in a horizontally split canvas. Tabbing order is based on the order in which you create controls having the style tabStop (see page 225).

To specify what percentage of the split canvas a control occupies when the split canvas is initially displayed, use ISplitCanvas::setSplitWindowPercentage (see page 828). The percentages you set determine the size of the window panes containing the controls, not the size of the controls themselves. Some testing might be necessary to determine the proper percentages to specify for each control to obtain the optimal initial appearance of the split canvas.

© Copyright IBM Corp. 1994

Note: If the split canvas has split bars and the user drags the split bars, the percentages you set will change. Your percentage values determine the initial appearance of the split canvas and the controls it contains. The minimum size to which a user can size a pane is the width of the split bar.

You can create a split canvas without split bars by using the style noSplitBars (see page 834). When you use this style, each pane maintains its size *ratio* to the other panes when users size the canvas. However, users cannot modify the percentage of the canvas occupied by a pane as they can when split bars are available.

When the sum of the controls' percentages is equal to 100, each window pane is given its requested percentage of the split canvas.

However, when the sum of the controls' percentages is less than or greater than 100, the percentages are converted to ratios. In this situation, the window panes containing the controls are sized on a one-to-one ratio, with each pane being given an equal size.

For example, if your split canvas has three controls and you specify no percentages, each window pane is given the ratio of 10:10:10. Thus, each pane gets roughly one-third of the window. The percentages are rounded off, so in this example the first window pane gets 34 percent of the split canvas and the other two panes get 33 percent.

If you set the percentages for either some of the controls or all of the controls, the total of the percentages (including any default percentages of 10) could still be less than or greater than 100. In this situation, the window panes that contain these controls are given a ratio determined by these percentages.

For example, suppose you have three controls, as in the preceding example, but this time you let only the first two controls take the default percentage of 10. If you set the third control to 60 percent, the sum of the percentages is 80 (10+10+60=80), which is less than 100. However, in this case, instead of a ratio of 10:10:10, the window panes containing the controls are given a ratio of 10:10:60. Therefore, the first pane gets 10/80 of the split canvas (13 percent), the second pane gets 10/80 (12 percent), and the third pane gets 60/80 (75 percent).

If you set percentages for all of your controls and the sum of those percentages is less than or greater than 100, the window panes containing the controls are given a ratio determined by the percentages that you set.

If a user changes the size of a split canvas, its window panes are sized to fill the split canvas by default. Each pane maintains its size ratio to the other panes in the canvas when this occurs.

Motif Information: Motif does not support the enumeration SplitBarEdge. The value of the edge is always zero. The enumerator splitBarMiddle specifies the thickness of the split bar.

Nested Classes

ISplitCanvas::Style

Constructors

```
public:
ISplitCanvas(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initialSize = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

You can construct objects of this class using the following parameters:

windowId Window identifier of the canvas you are constructing.

Note: For OS/2, this value should be limited to the range 0 through 65535.

It is recommended that you:

- Give unique identifiers to all windows in the same frame window. *In the same window* means that the frame window is the first frame in their parent window chain.
- Give the client window a window ID of FID_CLIENT (0x8008).
- Do not use the other FID_xxx values.

parent The parent window of the canvas you are constructing. You must specify a parent window. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0.

The parent window is primarily used for visible relationships.

owner Owner window of the canvas you construct.

Note: In OS/2, the owner window is primarily used for routing notification events and unprocessed messages. OS/2 also uses the owner window chain to inherit colors.

In Motif, the owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial (Optional) The initial position and size of the canvas you are constructing. The initial position is the lower-left corner of the canvas relative to the lower-left corner of the parent window.

style (Optional) The window's characteristics. These can be a combination of the ISplitCanvas::Style and the IWindow::Style.

Public Members

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

Returns the default style. The default style is classDefaultStyle (see page 834) unless you have changed the style using setDefaultStyle (see page 827).

orientation
```
Orientation orientation() const;
```

Returns an enumerator Orientation (see page 832) for canvas split bar orientation.

resetSplitBarEdgeColor
Condition: (1) Ignored on AIX

```
virtual ISplitCanvas &resetSplitBarEdgeColor();
```

Resets the color of the edge of the split bar by undoing a previous set.

resetSplitBarMiddleColor
Condition: (1) Ignored on AIX

```
virtual ISplitCanvas &resetSplitBarMiddleColor();
```

Resets the color of the middle of the split bar by undoing a previous set.

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

Sets the default style for all subsequent canvases.

style Use the styles provided by ISplitCanvas::Styles (see page 834) to specify the default style.

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

Treats the following as changes to the layout of its child windows:

- Changing a window size
- Adding a child window
- Removing a child window

setOrientation

```
virtual ISplitCanvas &setOrientation(Orientation value);
```

Sets the direction in which the split bar is drawn.

value Use the enumeration Orientation (see page 832) to specify the direction.

setSplitBarEdgeColor

Condition: (1) Ignored on AIX

```
virtual ISplitCanvas &setSplitBarEdgeColor(
                        const IColor &color);
```

Sets the color for the edge of the split bar in a split canvas.

setSplitBarMiddleColor

```
virtual ISplitCanvas &setSplitBarMiddleColor(
                        const IColor &color);
```

Sets the color for the middle of the split bar in a split canvas.

setSplitBarThickness

```
virtual ISplitCanvas &setSplitBarThickness(
                        SplitBarArea value,
                        unsigned long thickness);
```

Sets the thickness of the specified area of the split bar.

value Use the enumeration SplitBarArea (see page 833) to specify the area.

thickness Thickness of the edge or middle in pixels. If you do not call this function, the thickness is the width of the system's frame sizing border.

setSplitWindowPercentage

```
virtual ISplitCanvas &setSplitWindowPercentage(
                        IWindow *window,
                        unsigned long percentage);
```

Sets the percentage of the canvas occupied by the specified window. If you do not call this function, the split canvas gives the child window a default of 10 percent of its width and height. If the sum of the percentages of all child windows does not equal 100, this value is treated as a ratio.

Note: If the split canvas is already shown, this function does not update it immediately. You must call IWindow::refresh (see page 935) to update the appearance of the canvas.

window The child window you are indirectly sizing.

percentage
The portion of the split window that the child window will occupy.

splitBarEdgeColor

Condition: (1) Ignored on AIX

```
virtual IColor splitBarEdgeColor() const;
```

Returns the color of the edge of the split bar or the default if no color for the area has been set.

splitBarMiddleColor

```
virtual IColor splitBarMiddleColor() const;
```

Returns the color of the middle of the split bar or the default if no color for the area has been set.

splitBarThickness

```
unsigned long splitBarThickness(SplitBarArea value);
```

Returns the thickness of the specified area of the split bar.

value Use the enumeration SplitBarArea (see page 833) to specify the area of the split bar.

splitWindowPercentage

```
unsigned long splitWindowPercentage(IWindow *window);
```

Returns the percentage of the canvas currently occupied by the specified window.

window The child window.

Figure 285. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	ICanvas

Figure 285. Inherited Public Members

Member function	Class
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
defaultPushButton	ICanvas
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl

Figure 285. Inherited Public Members

Member function	Class
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isTabStop	ICanvas
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow

Figure 285. Inherited Public Members

Member function	Class
matchForMnemonic	ICanvas
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
origDefaultButtonHandle	ICanvas
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow

Figure 285. Inherited Public Members

Member function	Class
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ICanvas
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

layout

```
virtual ISplitCanvas &layout();
```

Computes the position and size of all child controls.

Figure 286. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
areChildrenReversed	ICanvas
calcMinimumSize	IWindow
calcMinimumSize	ICanvas
create	IWindow
defaultProcedure	IWindow

Figure 286. Inherited Protected Members

Member function	Class
deleteIsInProcess	IWindow
dispatch	IWindow
fixupChildren	ICanvas
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
layout	ICanvas
layoutSize	ICanvas
notificationHandler	IWindow

Figure 286. Inherited Protected Members

Member function	Class
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ICanvas
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	ICanvas
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setLayoutSize	ICanvas
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	ICanvas
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Orientation

```
public:
enum Orientation
  {
  horizontalSplit,
  verticalSplit,
  };
```

Use these enumerators to specify the direction of the split bar:

horizontalSplit
Draws the split bar horizontally, placing the panes top to bottom.

verticalSplit
Draws the split bar vertically, placing the panes left to right.

SplitBarArea

```
public:
enum SplitBarArea
  {
  splitBarEdge,
  splitBarMiddle,
  };
```

Use these enumerators to specify different areas of the split bar:

splitBarEdge
Specifies the top and bottom edges of a horizontal split bar or the left and right edges of a vertical split bar.

splitBarMiddle
Specifies the middle of the split bar.

Chapter 193. ISplitCanvas::Style

Derivation

IBase
  IBitFlag
    ISplitCanvas::Style

Header File: isplitcv.hpp

The nested class ISplitCanvas::Style provides a set of valid split canvas styles for the defaultStyle and setDefaultStyle functions, and for the constructor of the ISplitCanvas (see page 824) class. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (see page 70)
IWindow::Style (see page 957)

Styles

classDefaultStyle
Specifies the original default style for this class, which is vertical | IWindow::visible.

horizontal
Draws the split bar horizontally, placing the panes top to bottom.

Note: You cannot specify this style with vertical.

noSplitBars
Specifies no split bars.

vertical
Draws the split bar vertically, placing the panes left to right. This style is the default.

Note: You cannot specify this style with horizontal.

Public Members

Figure 287. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag

Figure 287. Inherited Public Members

Member function	Class
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Protected Members

Figure 288. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 194. IStaticText

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IStaticText
```

Header File: istattxt.hpp

Objects of the IStaticText class create and manage the static text control windows. Static text controls are simple text fields that do not accept user input. You can use the classes derived from IStaticText to display bit maps 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:

- IMouseClickHandler (see page 539)
- IPaintHandler (see page 647)
- IResizeHandler (see page 732)

Portability Considerations: OS/2 applications can use the IStaticText control to create boxes that label, box, or frame other controls.

The IStaticText control can be both parent/child and owner of these other controls. However, for portable applications, you should use IFrameWindow, IOutlineBox, or the canvas classes instead of IStaticText.

Presentation Manager Information: If you do not set a fill color, the background color is used as the fill color.

Nested Classes

IStaticText::Style

Constructors

```
public:
IStaticText(
   unsigned long id,
   IWindow *parent,
   IWindow *owner,
```


© Copyright IBM Corp. 1994

```
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));

public:
IStaticText(unsigned long id, IWindow *parent);

public:
IStaticText(const IWindowHandle &handle);
```

You can construct objects of this class by:

- Creating the specified static text control and an object for it
- Creating the object for the specified static text control

Public Members

alignment

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

Returns the current alignment for this static text object. The returned value is an Alignment (see page 845) enumerator.

backgroundColor

Condition: (2) OS/2 Only

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

Returns the background color of the static text object. If you do not set a color for this area, the default is returned.

Presentation Manager Information: The default is from the owner window chain or color scheme on OS/2.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 846) unless you have changed it using setDefaultStyle (see page 841).

disableFillBackground

```
virtual IStaticText &disableFillBackground();
```

Draws text over the current background.

Motif Information: The fillBackground style is always enabled in AIX; therefore, this function has no effect.

disableHalftone

```
virtual IStaticText &disableHalftone();
```

Removes the halftone (see page 846) from the text.

Motif Information: The halftone style is not available in AIX; therefore, this function has no effect.

disableStrikeout

```
virtual IStaticText &disableStrikeout();
```

Removes the line drawn through the text.

Motif Information: The strikeout style is not available in AIX; therefore, this function has no effect.

disableUnderscore

```
virtual IStaticText &disableUnderscore();
```

Removes the line beneath the text.

Motif Information: The underscore style is not available in AIX; therefore, this function has no effect.

enableFillBackground

```
virtual IStaticText &enableFillBackground(
                      Boolean enable = true);
```

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.

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 (see page 837).

Motif Information: The fillBackground style is ignored in AIX; therefore, this function has no effect.

enableHalftone

```
virtual IStaticText &enableHalftone(Boolean enable = true);
```

Draws the text using the style halftone (see page 846).

Motif Information: The halftone style is not available in AIX; therefore, this function has no effect.

enableStrikeout

```
virtual IStaticText &enableStrikeout(Boolean enable = true);
```

Draws a line through the text.

Motif Information: The strikeout style is not available in AIX; therefore, this function has no effect.

enableUnderscore

```
virtual IStaticText &enableUnderscore(
                        Boolean enable = true);
```

Draws a line beneath the text.

Motif Information: The underscore style is not available in AIX; therefore, this function has no effect.

fillColor

```
virtual IColor fillColor() const;
```

Returns the fill color value of the static text. If you do not set the fill color, the default is returned.

Motif Information: Fill color and background color are identical in the AIX environment. Therefore, this function returns the background color in this environment.

foregroundColor

Condition: (2) OS/2 Only

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

Returns the foreground color value of the static text. If you do not set the color for the area, the default is returned.

hasFillBackground

```
Boolean hasFillBackground() const;
```

If the background is erased before any text is drawn, true is returned.

Motif Information: The fillBackground style is ignored in AIX. This function returns false.

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

Queries whether the text has the style halftone (see page 846).

Motif Information: The halftone style is not available in AIX. This function returns false.

isStrikeout
```
Boolean isStrikeout() const;
```

Queries whether the text has the style strikeout (see page 846).

Motif Information: The strikeout style is not available in AIX. This function returns false.

isUnderscore
```
Boolean isUnderscore() const;
```

Queries whether the text has the style underscore (see page 846).

Motif Information: The underscore style is not available in AIX. This function returns false.

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

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.

moveSizeTo
Condition: (3) AIX only

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

Changes the position and size of the static text window using a lower-left origin based coordinate system.

Motif Information: This member function overrides the IWindow version for Motif. It obtains the proper vertical alignment for static text.

resetFillColor
```
virtual IStaticText &resetFillColor();
```

Resets the fill color by undoing a previous set.

Motif Information: This function has no effect in the AIX environment.

setAlignment

```
virtual IStaticText &setAlignment(
                      Alignment alignment = IStaticText::topLeft);
```

Sets the alignment of the static text.

alignment Use the enumerators provided by Alignment (see page 845) to specify the alignment of the static text.

setDefaultStyle

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

Sets the default style for all subsequent static text controls.

style Use the styles provided by IStaticText::Styles (see page 846) to specify the default style.

setFillColor

```
virtual IStaticText &setFillColor(const IColor &color);
```

Sets the color for the area of the static text control window that does not have text.

Motif Information: This function is equivalent to setBackgroundColor in the AIX environment.

setLimit

```
virtual IStaticText &setLimit(unsigned long limit = 0);
```

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 (see page 768) or IMultiCellCanvas (see page 541) to avoid causing the canvas to update the layout of its child windows with each change to the static text contents.

calcLimitSize (see page 844) 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:

- The calculated miminum 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.

limit If you use the default of 0, the minimum size will be what is needed to display the current text.

setText

```
virtual IStaticText &setText(const IResourceId &text);

virtual IStaticText &setText(const char *text);
```

Sets the text for the control. If appropriate, this function also notifies a parent canvas to update the layout for its children.

Figure 289. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow

Figure 289. Inherited Public Members

Member function	Class
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow

Figure 289. Inherited Public Members

Member function	Class
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow

Figure 289. Inherited Public Members

Member function	Class
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcLimitSize

```
virtual ISize calcLimitSize() const;
```

Returns a size based on the text limit value and the current font.

calcMinimumSize

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

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 (see page 844)

passEventToOwner

Condition: (3) AIX only

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

Determines if the event is passed on to the owner.

Figure 290. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier

Figure 290. Inherited Protected Members

Member function	Class
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Enumerations

Alignment

```
public:
enum 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 Information: In AIX, topLeftWrapped is equivalent to the enumerator topLeft. Use topLeft alignment whenever the static text contains new-line separators.

Chapter 195. IStaticText::Style

Derivation

IBase
  IBitFlag
    IStaticText::Style

Header File: istattxt.hpp

The nested class IStaticText::Style provides a set of static text styles for the static member functions of the class IStaticText (see page 836). You can use these styles with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)

Styles

bottom
Aligns the text at the bottom of the window.

center
Centers the text.

classDefaultStyle
Specifies the original default style for this class, which is left | top | fillBackground | IWindow::visible.

fillBackground
Erases the background, using the currently set fill color, before drawing the text.

halftone
Draws the text in halftone color.

left
Left-justifies the text.

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.

right
Right-justifies the text.

strikeout
Draws the text with a line through it.

© Copyright IBM Corp. 1994

top
Aligns the text at the top of the window.

underscore
Underlines the text.

vertCenter
Aligns the text vertically in the center of the window.

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.

Note: In OS/2 Presentation Manager, when you use static text objects in a multiple-cell canvas and your static text requires multiple-lines, you must override calcMinimumSize to return a rectangle for more than one line of text.

AIX supports multiple-line text only with explicit new-line characters.

Portability Considerations: AIX does not support several styles. You can use these styles for portable applications, but AIX will ignore these styles.

Motif Information: 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 multiple-line text by explicit use of new-line characters.

AIX does not support the following styles:

- fillBackground
- halftone
- mnemonic
- strikeout
- underscore
- wordBreak

Public Members

Figure 291. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*

Figure 291. Inherited Public Members

Member function	Class
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 292. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 196. IStringHandle

Derivation

IBase
  IHandle
    IStringHandle

Header File: ihandle.hpp

Objects of the IStringHandle class access a string.

Presentation Manager Information: IStringHandle is an alias for the OS/2 Programmer's Toolkit type HSTR.

Constructors

```
public:
IStringHandle(Value hstr = 0);
```

You can construct objects of this class from a string handle (a value of type IHandle::Value), which defaults to 0.

Public Members

operator Value

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

Returns the IStringHandle value.

Figure 293. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 197. ISubmenu

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IMenu
          ISubmenu
```

Header File: isubmenu.hpp

Objects of the ISubmenu class create a submenu object from a menu handle. This class overrides functions of IMenu (see page 479) to provide undoable menu item functions. The menu item functions are intended to act upon a submenu. Any changes are undone when the menu ends.

Motif Information: The User Interface Class Library adds an X/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

Nested Classes

ISubmenu::Cursor

Constructors

```
public:
ISubmenu(const IWindowHandle &submenuHandle);
```

You can only construct objects of this class from a window handle of the submenu window. Typically, only menu handlers create submenus.

Public Members

add

```
virtual ISubmenu &add(IMenuItem &menuItem, Cursor &cursor);
```

Adds the specified menu item at the position of the specified cursor by pushing down everything after the cursor.

© Copyright IBM Corp. 1994

addAsNext

```
virtual ISubmenu &addAsNext(
                    IMenuItem &menuItem,
                    Cursor &cursor);
```

Adds the specified menu item as the next item in the submenu and sets the specified cursor there.

addAt

```
virtual ISubmenu &addAt(
                    IMenuItem &menuItem,
                    Cursor &cursor);
```

Adds the specified menu item at the position of the specified cursor by pushing down everything after the cursor.

Note: This function is obsolete. Use IMenu::add instead.

addBitmap

```
virtual ISubmenu &addBitmap(
                    unsigned long newItemId,
                    const IBitmapHandle &itemBitmap,
                    unsigned long intoSubmenuId = 0);

virtual ISubmenu &addBitmap(
                    unsigned long newItemId,
                    const IResourceId &newItemBitmapResId,
                    unsigned long intoSubmenuId = 0);

virtual ISubmenu &addBitmap(
                    unsigned long newItemId,
                    unsigned long newItemBitmapResId,
                    unsigned long intoSubmenuId = 0);
```

Adds a bit-map menu item as the last item in a menu or submenu. The parameters are the following:

newItemId The new menu item to add to a menu or submenu.

bitmapResId
: The bit map to load for the button face of the new menu item. This parameter accepts the resource ID of a bit map to load.

itemBitmap
: The bit map to load for the button face of the new menu item. This parameter accepts a currently available bit-map handle.

intoSubmenuID
: Not used. It is only present to allow this function to override the function inherited from IMenu.

Motif Information: 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

```
virtual ISubmenu &addItem(
                    IMenuItem &menuItem,
                    unsigned long intoSubmenuId = 0);
```

Adds a menu item represented by an IMenuItem (see page 512) 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.

Motif Information: 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

addNextAt

```
virtual ISubmenu &addNextAt(
                    IMenuItem &menuItem,
                    Cursor &cursor);
```

Adds the specified menu item as the next item in the submenu and sets the specified cursor there.

Note: This function is obsolete. Use IMenu::addAsNext instead.

addSeparator

```
virtual ISubmenu &addSeparator(unsigned long newItemId = 0);

virtual ISubmenu &addSeparator(
                    unsigned long newItemId,
                    unsigned long intoSubmenuId);
```

Adds a separator menu item with the specified ID as the last item in a submenu.

Note: Unlike IMenu::addSeparator, the single parameter version of this function accepts an optional *newItemId*.

addText

```
virtual ISubmenu &addText(
                    unsigned long newItemId,
                    const char *itemText,
                    unsigned long intoSubmenuId = 0);

virtual ISubmenu &addText(
                    unsigned long newItemId,
                    const IResourceId &newItemTextResId,
                    unsigned long intoSubmenuId = 0);
```

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.

Motif Information: 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

checkItem

```
virtual ISubmenu &checkItem(
                    unsigned long itemId,
                    Boolean checked = true);
```

Places a select state indicator to the left of the specified item.

Motif Information: The select state indicator is a square-shaped object to the left of the specified item.

Presentation Manager Information: The select state indicator is a check mark in front of the menu item.

cursor

```
Cursor cursor(unsigned long itemId) const;
```

Returns an IMenu::Cursor (see page 495) object and sets the cursor to point to the specified menu item.

deleteAt

```
virtual ISubmenu &deleteAt(Cursor &cursor);
```

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.

deleteItem

```
virtual ISubmenu &deleteItem(unsigned long itemId);
```

Removes the specified item from a menu.

disableItem

```
virtual ISubmenu &disableItem(unsigned long itemId);
```

Makes the specified menu item unselectable.

elementAt

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

Returns the menu item at the position of the specified cursor.

enableItem

```
virtual ISubmenu &enableItem(
                    unsigned long itemId,
                    Boolean enabled = true);
```

Makes the menu item selectable.

handle

```
virtual IWindowHandle handle() const;
```

Returns the submenu window handle.

removeSubmenu

```
virtual ISubmenu &removeSubmenu(
                    unsigned long itemWithSubmenuId);
```

Removes the menu item submenu indicator and destroys the submenu. Typically, you use this member function to remove cascading menus.

removeSubmenuAt

```
virtual ISubmenu &removeSubmenuAt(Cursor &cursor);
```

Removes the menu item at the cursor position submenu indicator and destroys the submenu. Typically, you use this member function to remove cascading menus.

setBitmap

```
virtual ISubmenu &setBitmap(
                    unsigned long menuItemId,
                    const IResourceId &newBitmapResId);

virtual ISubmenu &setBitmap(
                    unsigned long menuItemId,
                    const IBitmapHandle &bitmapHandle);

virtual ISubmenu &setBitmap(
                    unsigned long menuItemId,
                    unsigned long newBitmapResId);
```

Replaces a specified menu item with a bit map.

setItem

```
virtual ISubmenu &setItem(const IMenuItem &menuItem);
```

Replaces a specified menu item's style and representation.

setText

```
virtual ISubmenu &setText(
                      unsigned long menuItemId,
                      const char *newText);

virtual ISubmenu &setText(
                      unsigned long menuItemId,
                      const IResourceId &newTextResId);
```

Replaces a specified menu item's text.

uncheckItem

```
virtual ISubmenu &uncheckItem(unsigned long itemId);
```

Removes the select state indicator from the left of the specified item.

Motif Information: The select state indicator is a square-shaped object to the left of the specified item.

Presentation Manager Information: The select state indicator is a check mark in front of the menu item.

undo

```
virtual ISubmenu &undo();
```

Reverses all the changes made to the menu using the ISubmenu member functions. You do not need to call this function because IMenuHandler (see page 508) automatically calls it.

Figure 294. Inherited Public Members

Member function	Class
activeColor	IWindow
add	IMenu
addAsNext	IMenu
addAt	IMenu
addBitmap	IMenu
addItem	IMenu
addNextAt	IMenu
addSeparator	IMenu
addSubmenu	IMenu
addText	IMenu
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow

Figure 294. Inherited Public Members

Member function	Class
backgroundColor	IMenu
borderColor	IWindow
characterSize	IWindow
checkItem	IMenu
childAt	IWindow
cursor	IMenu
defaultPushButton	IWindow
deleteAt	IMenu
deleteItem	IMenu
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IMenu
disabledForegroundColor	IWindow
disabledForegroundColor	IMenu
disableItem	IMenu
disableNotification	INotifier
disableNotification	IWindow

Figure 294. Inherited Public Members

Member function	Class
disableUpdate	IWindow
elementAt	IMenu
enable	IWindow
enableItem	IMenu
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IMenu
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IMenu
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IMenu
hiliteForegroundColor	IWindow
hiliteForegroundColor	IMenu
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isItemChecked	IMenu
isItemDisabled	IMenu
isItemEnabled	IMenu
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVerticalFlip	IMenu
isVisible	IWindow
itemProvider	IWindow

Figure 294. Inherited Public Members

Member function	Class
itemRect	IMenu
layoutAdjustment	IWindow
matchForMnemonic	IWindow
menuItem	IMenu
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
numberOfItems	IMenu
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeConditionalCascade	IMenu
removeSubmenu	IMenu
removeSubmenuAt	IMenu
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBackgroundColor	IMenu
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledBackgroundColor	IMenu
resetDisabledForegroundColor	IWindow
resetDisabledForegroundColor	IMenu
resetForegroundColor	IWindow
resetForegroundColor	IMenu
resetHiliteBackgroundColor	IWindow
resetHiliteBackgroundColor	IMenu
resetHiliteForegroundColor	IWindow

Figure 294. Inherited Public Members

Member function	Class
resetHiliteForegroundColor	IMenu
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
selectItem	IMenu
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBackgroundColor	IMenu
setBitmap	IMenu
setBorderColor	IWindow
setConditionalCascade	IMenu
setDisabledBackgroundColor	IWindow
setDisabledBackgroundColor	IMenu
setDisabledForegroundColor	IWindow
setDisabledForegroundColor	IMenu
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setForegroundColor	IMenu
setHiliteBackgroundColor	IWindow
setHiliteBackgroundColor	IMenu
setHiliteForegroundColor	IWindow
setHiliteForegroundColor	IMenu
setId	IWindow
setInactiveColor	IWindow
setItem	IMenu
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setSubmenu	IMenu
setText	IMenu
setVerticalFlip	IMenu
shadowColor	IWindow
show	IWindow

Figure 294. Inherited Public Members

Member function	Class
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
uncheckItem	IMenu
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 295. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 198. ISubmenu::Cursor

Derivation

IBase
  IVBase
    ISubmenu::Cursor

Header File: isubmenu.hpp

Objects of the ISubmenu::Cursor class create and manage the cursor for menu items.

Constructors

```
public:
Cursor(const ISubmenu &menu);
```

You can only construct objects of this class from an object of the class ISubmenu (see page 850).

Public Members

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

Flags this cursor as invalid.

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

Queries whether this cursor points to a valid menu item.

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

Points to the first menu item.

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

Points to the last menu item.

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

Points to the next menu item. If there is none, the cursor is invalidated.

© Copyright IBM Corp. 1994

setToPrevious

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

Points to the previous menu item. If there is none, the cursor is invalidated.

Figure 296. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 199. ISWP

Derivation

IBase
  ISWP

Header File: iswp.hpp

The ISWP class is a wrapper for the OS/2 Presentation Manager Toolkit Set Window Position (SWP) structure. This class' member functions access and set the various elements of this structure. In doing so, these functions get and set the window position and size. The accessor member functions get the window handle (hwnd) as an IWindowHandle (see page 960) and return the flags as an unsigned long.

The window position structure contains the following elements:

flags	A set of window size and move flags. The Presentation Manager Toolkit SWP_* constants define the flags.
position	A window position represented by an IPoint (see page 655).
size	A window size represented by an ISize (see page 790).
behind	The window handle identifying the window to place the window associated with this structure behind.
hwnd	The window handle of the window associated with this structure.

Motif Information: AIX does not support this class.

Public Members

flags
```
unsigned long &flags();
```

Returns the flags stored in this object.

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

Returns the window position stored in this object.

setPosition
```
ISWP &setPosition(const IPoint &aPoint);
```

Sets the window position.

© Copyright IBM Corp. 1994

setSize
```
ISWP &setSize(const ISize &aSize);
```

Sets the window size.

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

Returns the window size stored in this object.

windowHandle
```
IWindowHandle windowHandle() const;
```

Returns the handle of the associated window.

Figure 297. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 200. ISWPArray

Derivation

IBase
 ISWPArray

Header File: iswp.hpp

The ISWPArray class is a wrapper for an array of OS/2 Presentation Manager Toolkit SWP structures. This class treats these structures as ISWP objects and provides functions to:

- Access a given element of the array
- Locate the element corresponding to a specified window ID

Motif Information: AIX does not support this class.

Constructors

```
public:
ISWPArray(ISWP *array, unsigned dimension);
```

You can construct objects of this class by providing:

array A pointer to an ISWP object. This pointer is presumed to be the first element of the array. Typically, you create an ISWP* by casting a Presentation Manager PSWP pointer.

dimension The number of elements in the array.

Public Members

indexOf

```
unsigned indexOf(unsigned long id) const;
```

Returns a reference to the element corresponding to a window.

id The ID of the window corresponding to the element.

operator []

```
ISWP &operator [](unsigned index);
```

Returns a reference to the specified element of the array.

index The element index of the SWP structure in the array.

© Copyright IBM Corp. 1994

size

```
unsigned size() const;
```

Returns the dimension of the array.

Figure 298. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
operator <<	IBase*

Note: * Application Support Class Library

Chapter 201. ISystemBitmapHandle

Derivation

IBase
  IHandle
    IBitmapHandle
      ISystemBitmapHandle

Header File: ihandle.hpp

Objects of the ISystemBitmapHandle class access system bit-map resources. A system bit map is a special type of bit map that is not loaded from a resource file.

Constructors

Condition: (2) OS/2 Only

```
public:
ISystemBitmapHandle(Identifier bitmapId);
```

You can construct objects of this class from an identifier for the specific system bit-map resource you want to associate with this handle. The enumeration Identifier (see page 866) provides the valid set of these resource identifiers.

Public Members

operator Value

Condition: (2) OS/2 Only

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

Returns the ISystemBitmapHandle value.

Figure 299. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*
operator =	IBitmapHandle
operator Value	IBitmapHandle

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Enumerations

Identifier

Condition: (2) OS/2 Only

```
public:
enum 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,
  };
```

These enumerators specify system bit-map resources.

Chapter 202. ISystemMenu

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IMenu
          ISystemMenu
```

Header File: isysmenu.hpp

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 (see page 479) and ISubmenu (see page 850) to change the ISystemMenu. You can also add objects of the class IMenuItem (see page 512) to the ISystemMenu.

Use the following constants when calling IMenu member functions to affect items on the system menu:

```
static const unsigned long
   idPulldown,
   idRestore,
   idMove,
   idSize,
   idMinimize,
   idMaximize,
   idHide,
   idClose,
   idWindowList;
```

When you want to add to the system menu pull-down, specify the idPulldown constant for *intoSubmenuId* when calling IMenu::addItem. For example:

```
sysmenu.addItem(myMenuItem, ISystemMenu::idPulldown);
```

If you do not specify 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 bit-map items rather than text.

Note: When adding a bit-map item to the title bar using addBitmap, myBitmapHandle should reference a bit map whose dimensions are 40 x 36. For example:

```
sysmenu.addBitmap(myMenuItem, myBitmapHandle);
```

Additionally, if you add more that one bit map to the right of the system menu icon, draw each bit map so that its left edge provides a visual separator.

The member functions IMenu::disableItem (see page 485) and ISubmenu::disableItem (see page 853) can have no effect in certain situations. For example, you cannot

© Copyright IBM Corp. 1994

disable the Size menu item on the system menu if the frame window has a sizing border.

Portability Considerations: AIX does not support the ISystemMenu::idClose constant. 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. Within systemCommand, compare ISystemMenu::idClose to the event's command ID. This is useful when you require a close down confirmation window.

Motif Information: AIX does not support this class.

Constructors

Condition: (1) Ignored on AIX

```
public:
ISystemMenu(IFrameWindow *owner);
```

You can only construct objects of this class from an owner frame window.

Public Members

Figure 300. Inherited Public Members

Member function	Class
activeColor	IWindow
add	IMenu
addAsNext	IMenu
addAt	IMenu
addBitmap	IMenu
addItem	IMenu
addNextAt	IMenu
addSeparator	IMenu
addSubmenu	IMenu
addText	IMenu
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow

Figure 300. Inherited Public Members

Member function	Class
backgroundColor	IMenu
borderColor	IWindow
characterSize	IWindow
checkItem	IMenu
childAt	IWindow
cursor	IMenu
defaultPushButton	IWindow
deleteAt	IMenu
deleteItem	IMenu
disable	IWindow
disabledBackgroundColor	IWindow
disabledBackgroundColor	IMenu
disabledForegroundColor	IWindow
disabledForegroundColor	IMenu
disableItem	IMenu
disableNotification	INotifier
disableNotification	IWindow

Figure 300. Inherited Public Members

Member function	Class
disableUpdate	IWindow
elementAt	IMenu
enable	IWindow
enableItem	IMenu
enableNotification	INotifier
enableNotification	IWindow
enableNotification	IMenu
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IMenu
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteBackgroundColor	IMenu
hiliteForegroundColor	IWindow
hiliteForegroundColor	IMenu
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isItemChecked	IMenu
isItemDisabled	IMenu
isItemEnabled	IMenu
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isValid	IWindow
isVerticalFlip	IMenu
isVisible	IWindow
itemProvider	IWindow

Figure 300. Inherited Public Members

Member function	Class
itemRect	IMenu
layoutAdjustment	IWindow
matchForMnemonic	IWindow
menuItem	IMenu
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
numberOfItems	IMenu
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeConditionalCascade	IMenu
removeSubmenu	IMenu
removeSubmenuAt	IMenu
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBackgroundColor	IMenu
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledBackgroundColor	IMenu
resetDisabledForegroundColor	IWindow
resetDisabledForegroundColor	IMenu
resetForegroundColor	IWindow
resetForegroundColor	IMenu
resetHiliteBackgroundColor	IWindow
resetHiliteBackgroundColor	IMenu
resetHiliteForegroundColor	IWindow

Figure 300. Inherited Public Members

Member function	Class
resetHiliteForegroundColor	IMenu
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
selectItem	IMenu
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBackgroundColor	IMenu
setBitmap	IMenu
setBorderColor	IWindow
setConditionalCascade	IMenu
setDisabledBackgroundColor	IWindow
setDisabledBackgroundColor	IMenu
setDisabledForegroundColor	IWindow
setDisabledForegroundColor	IMenu
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setForegroundColor	IMenu
setHiliteBackgroundColor	IWindow
setHiliteBackgroundColor	IMenu
setHiliteForegroundColor	IWindow
setHiliteForegroundColor	IMenu
setId	IWindow
setInactiveColor	IWindow
setItem	IMenu
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setSubmenu	IMenu
setText	IMenu
setVerticalFlip	IMenu
shadowColor	IWindow
show	IWindow

Figure 300. Inherited Public Members

Member function	Class
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
uncheckItem	IMenu
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 301. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 203. ISystemPointerHandle

Derivation

IBase
  IHandle
    IPointerHandle
      ISystemPointerHandle

Header File: ihandle.hpp

Objects of the ISystemPointerHandle class access system pointer resources. A system pointer is a special type of pointer that is not loaded from a resource file.

Constructors

Condition: (2) OS/2 Only

```
public:
ISystemPointerHandle(
   Identifier pointerId,
   Boolean makeCopy = false);
```

You can construct objects of this class from an identifier for the specific system-pointer resource you want to to associate with this handle. The enumeration Identifier (see page 873) 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.

Public Members

operator Value

Condition: (2) OS/2 Only

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

Returns the ISystemPointerHandle value.

Figure 302. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Figure 302. Inherited Public Members

Member function	Class
operator =	IPointerHandle
operator Value	IPointerHandle

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Enumerations

Identifier

Condition: (2) OS/2 Only

```
public:
enum Identifier
  {
  arrow,
  text,
  wait,
  move,
  sizeNWSE,
  sizeNESW,
  sizeHorizontal,
  sizeVertical,
  standardApplication,
  information,
  question,
  error,
  warning,
  illegalOperation,
  singleFile,
  multipleFile,
  folder,
  program,
  dragFade,
  };
```

The preceding enumerators specify various system pointers.

Chapter 204. ITextControl

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl

Header File: itextctl.hpp

The ITextControl class is an abstract base class for all text control window classes. It provides functions that manipulate text strings.

Motif Information: ITextControl is not the class corresponding to a text widget. If you are looking for a class for a text widget, please read about IEntryField (see page 256) and IMultiLineEdit (see page 557).

Constructors

```
protected:
ITextControl();
```

Because this class is an abstract base class, you cannot construct objects of this class.

Public Members

clipboardHasTextFormat

```
static Boolean clipboardHasTextFormat();
```

Returns true if the clipboard is in text format.

Motif Information: If one of the registered formats for the clipboard data is of type "STRING," this function returns true.

displaySize

```
virtual ISize displaySize(const char *text = 0) const;
```

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.

© Copyright IBM Corp. 1994

Motif Information: This functions uses the underlying widget's current font list. It supports multiple-byte character sets but not sheared or rotated strings.

Presentation Manager Information: 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.

setLayoutDistorted

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

Processes a font change like a minimum size change. This is a virtual override of setLayoutDistorted (see page 941).

setText

```
virtual ITextControl &setText(const IResourceId &text);

virtual ITextControl &setText(const char *text);
```

Sets the control window's text.

text

```
virtual IString text() const;
```

Returns the control window's text.

textLength

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

Returns the current length of the control window's text in bytes.

Figure 303. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow

Figure 303. Inherited Public Members

Member function	Class
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow

Figure 303. Inherited Public Members

Member function	Class
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow

Figure 303. Inherited Public Members

Member function	Class
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setMinimumSize	IWindow
setOwner	IWindow

Figure 303. Inherited Public Members

Member function	Class
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 304. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow

Figure 304. Inherited Protected Members

Member function	Class
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 205. ITextSpinButton

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          IBaseSpinButton
            ITextSpinButton
```

Header File: ispintxt.hpp

Objects of the ITextSpinButton class create and manage 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 (see page 247)
IEditVerifyHandler (see page 253)
IFocusHandler (see page 307)
IKeyboardHandler (see page 455)
IMouseClickHandler (see page 539)
IResizeHandler (see page 732)
ISpinHandler (see page 821)
IPaintHandler (see page 647)

Portability Considerations: 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 style.

Motif Information: The ITextSpinButton constructor creates objects of this class using several 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 ITextSpinButton object via private callbacks and a default handler. The ITextSpinButton class uses a default handler derived from the class IKeyboardHandler. Therefore, you should 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 prior to the default handler.

© Copyright IBM Corp. 1994

Nested Classes

ITextSpinButton::Cursor
ITextSpinButton::Style

Constructors

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

Condition: (2) OS/2 Only

```
public:
ITextSpinButton(unsigned long id, IWindow *parent);
```

Condition: (2) OS/2 Only

```
public:
ITextSpinButton(const IWindowHandle &handle);
```

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 (Optional) A rectangle defining the size and placement of the text spin button window.

  style (Optional) The style of the control.

- 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.

- Creating the object for the specified text spin button control

  handle The window handle of an existing text spin button control.

Public Members

add

```
virtual ITextSpinButton &add(
                            const IResourceId &item,
                            unsigned long index);

virtual ITextSpinButton &add(
                            const char *string,
                            Cursor &cursor);

virtual ITextSpinButton &add(
                            const IResourceId &item,
                            Cursor &cursor);

virtual ITextSpinButton &add(
                            const char *string,
                            unsigned long index);

virtual ITextSpinButton &add(
                            const char * const *stringArray,
                            unsigned long index,
                            unsigned long count);
```

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.

addAsFirst

```
virtual ITextSpinButton &addAsFirst(const char *string);

virtual ITextSpinButton &addAsFirst(
                            const IResourceId &item);
```

Adds the item as the first item. You can use this function to add an item to an empty ITextSpinButton object.

addAsLast

```
virtual ITextSpinButton &addAsLast(const char *string);

virtual ITextSpinButton &addAsLast(const IResourceId &item);
```

Adds the item as the last item. You can use this function to add an item to an empty text ITextSpinButton object.

addAsNext

```
virtual ITextSpinButton &addAsNext(
                            const IResourceId &item,
                            Cursor &cursor);

virtual ITextSpinButton &addAsNext(
                            const char *string,
                            Cursor &cursor);
```

Adds the item following the current cursor position and sets the cursor on it.

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 888) unless you have changed the style using setDefaultStyle (see page 881).

elementAt

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

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

Returns the string at the cursor or index position.

isSpinFieldValid

```
virtual Boolean isSpinFieldValid(
                  Boolean caseSensitive = false) const;
```

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.

removeAll

```
virtual ITextSpinButton &removeAll();
```

Removes all items in the spin button.

removeAt

```
virtual ITextSpinButton &removeAt(Cursor &cursor);
```

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.

replaceAt

```
virtual ITextSpinButton &replaceAt(
                           const char *newString,
                           Cursor &cursor);

virtual ITextSpinButton &replaceAt(
                           const IResourceId &item,
                           Cursor &cursor);
```

Replaces the item at the cursor position.

setDefaultStyle

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

Sets the default style for all subsequent text spin buttons.

style Use the styles provided by ITextSpinButton::Styles (see page 888) to specify the default style.

setLimit

Condition: (3) AIX only

```
virtual ITextSpinButton &setLimit(unsigned long aNumber);
```

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.

setText

```
virtual ITextSpinButton &setText(const IResourceId &item);

virtual ITextSpinButton &setText(const char *string);
```

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.

spinDown

Condition: (3) AIX only

```
virtual ITextSpinButton &spinDown(unsigned long spinBy = 1);
```

Spins the button down the specified number of times.

spinTo

```
virtual ITextSpinButton &spinTo(unsigned long index);

virtual ITextSpinButton &spinTo(const Cursor &cursor);

virtual ITextSpinButton &spinTo(
                          const char *string,
                          Boolean caseSensitive = false);
```

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.

spinUp

Condition: (3) AIX only

```
virtual ITextSpinButton &spinUp(unsigned long spinBy = 1);
```

Spins the button up the specified number of times.

text

```
virtual IString text() const;
```

Returns the displayed contents of the spin field.

Figure 305. Inherited Public Members

Member function	Class
activeColor	IWindow
addBorder	IBaseSpinButton
alignment	IBaseSpinButton
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disableDataUpdate	IBaseSpinButton
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableFastSpin	IBaseSpinButton
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enable	IBaseSpinButton
enableDataUpdate	IBaseSpinButton
enableFastSpin	IBaseSpinButton
enableGroup	IControl
enableNotification	INotifier

Figure 305. Inherited Public Members

Member function	Class
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
foregroundColor	IBaseSpinButton
handle	IWindow
handleException	IWindow
hasBorder	IBaseSpinButton
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFastSpinEnabled	IBaseSpinButton
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isMaster	IBaseSpinButton
isServant	IBaseSpinButton
isShowing	IWindow

Figure 305. Inherited Public Members

Member function	Class
isSpinFieldValid	IBaseSpinButton
isTabStop	IWindow
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
isWriteable	IBaseSpinButton
itemProvider	IWindow
layoutAdjustment	IWindow
limit	IBaseSpinButton
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveSizeTo	IBaseSpinButton
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
removeBorder	IBaseSpinButton
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow

Figure 305. Inherited Public Members

Member function	Class
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAlignment	IBaseSpinButton
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLimit	IBaseSpinButton
setMaster	IBaseSpinButton
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
spinDown	IBaseSpinButton
spinUp	IBaseSpinButton
topHandle	IWindow
topHandle	IBaseSpinButton
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 306. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
calcMinimumSize	IBaseSpinButton
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
initialize	IBaseSpinButton
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	IBaseSpinButton
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	IBaseSpinButton
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 206. ITextSpinButton::Cursor

Derivation

IBase
  IVBase
    ITextSpinButton::Cursor

Header File: ispintxt.hpp

Objects of the ITextSpinButton::Cursor class create and manage the cursor for an ITextSpinButton (see page 878) 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.

Constructors

```
public:
Cursor(const ITextSpinButton &spinButton);
```

You can only construct objects of this class from an object of ITextSpinButton (see page 878).

Public Members

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

Flags this cursor as invalid.

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

Queries whether this cursor points to a valid item.

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

Points to the first item in the text list.

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

Points to the last item in the text list.

© Copyright IBM Corp. 1994

setToNext

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

Points to the next item in the text list. If there is none, the cursor is invalidated.

setToPrevious

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

Points to the previous item in the text list. If there is none, the cursor is invalidated.

Figure 307. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 207. ITextSpinButton::Style

Derivation

IBase
  IBitFlag
    ITextSpinButton::Style

Header File: ispintxt.hpp

The nested class ITextSpinButton::Style provides a set of text spin button styles for the static member functions of the class ITextSpinButton (see page 878). You can use this style in conjunction with the styles defined by the following nested classes:

IWindow::Style (see page 957)
IControl::Style (see page 225)
IBaseSpinButton::Style (see page 40)

Styles

classDefaultStyle
Specifies the original default style for this class, which is IBaseSpinButton::classDefaultStyle.

Public Members

Figure 308. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

© Copyright IBM Corp. 1994

Figure 309. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 208. IThread

Derivation

IBase
  IVBase
    IThread

Header File: ithread.hpp

The IThread class' objects represent threads of execution within the current program. This class provides support for implementing multithreaded programs. IThread objects provide access to all the tasking APIs of the operating system. In addition, these objects serve as the anchor for thread-specific information.

Generally, you use IThread objects 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 (see page 892).
- To create additional threads of execution by instantiating new IThread objects and starting them.
- To manipulate threads of execution initiated using alternate means, and for which only the thread's ID is known.

Portability Considerations: 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 with no adverse side effects. Therefore, they have been left in the interface for ease of porting applications to multiple platforms.

Motif Information: 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 IApplication with the call to IApplication::current().run(). 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 at this time.

© Copyright IBM Corp. 1994

Presentation Manager Information: Compile multithreaded programs (those that call the IThread::start member function) with /Gm+ in order to avoid unresolved externals at link time.

Nested Classes

IThread::Cursor

Constructors

```
public:
IThread();
```

```
public:
IThread(const IThread &aThread);
```

```
public:
IThread(const IThreadId &anID);
```

Condition: (2) OS/2 Only

```
public:
IThread(
   const IReference < IThreadFn > &aFnObjRef,
   Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

Condition: (2) OS/2 Only

```
public:
IThread(
   OptlinkFnPtr pfn,
   void *anArg,
   Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

Condition: (2) OS/2 Only

```
public:
IThread(
   SystemFnPtr pfn,
   unsigned long anArg,
   Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

You can construct objects of this class in the following ways:

- Using the default constructor. Use this form of the constructor to build an IThread to subsequently start using IThread::start (see page 897).
- With specification of code to run. Use this form of the constructor to construct a new IThread and immediately dispatch it. It is equivalent to using the default constructor and then dispatching the thread using IThread::start.

  This constructor permits you to specify whether or not the new thread is to be a GUI thread. If it is, ICurrentThread::initializeGUI (see page 235) is called automatically after the thread is started.

- From the thread ID of a previously started thread. Use this constructor to provide IThread functionality to threads created using alternate means (for example, native operating system system calls).

IThread has a virtual destructor that handles deallocating resources.

Note: The destructor does not stop the thread.

Public Members

adjustPriority

Condition: (1) Ignored on AIX

```
virtual IThread &adjustPriority(int delta);
```

Changes the thread priority level by a specified amount.

Motif Information: You cannot adjust the thread's priority on AIX.

Presentation Manager Information: The delta value is between -31 and 31.

asDebugInfo

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

Returns a representation of the thread as diagnostic information.

asString

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

Returns a string of the form "IThread(tid)."

autoInitGUI

```
virtual Boolean autoInitGUI() const;
```

Obtains the setting of the flag that automatically initializes the GUI for this thread.

Presentation Manager Information: Obtains the setting of the flag that automatically initializes Presentation Manager for this thread.

autoInitPM

```
virtual Boolean autoInitPM() const;
```

Obtains the setting of the flag that automatically initializes OS/2 Presentaion Manager for this thread.

Note: This function is obsolete. Use IThread::autoInitGUI instead.

current

```
static ICurrentThread &current();
```

Returns a reference to a thread object that represents the currently executing thread.

currentId

```
static IThreadId currentId();
```

Returns the current thread ID.

Motif Information: The AIX release of the User Interface Class Library only supports a single thread. Therefore, the current thread ID is always 1.

defaultAutoInitGUI

```
static Boolean defaultAutoInitGUI();
```

Obtains the default flag that automatically initializes the GUI. Unless this flag is explicitly set using IThread::setDefaultAutoInitGUI (see page 895), this function returns true for programs running in a GUI session and false for programs running in a non-GUI session.

defaultAutoInitPM

```
static Boolean defaultAutoInitPM();
```

Obtains the default flag that automatically initializes OS/2 Presentation Manager. Unless this flag is explicitly set using IThread::setDefaultAutoInitPM (see page 896), this function returns the following:

- true for programs running in a Presentation Manager session
- false for programs running in a non/Presentation Manager session

Note: This function is obsolete. Use IThread::defaultAutoInitGUI instead.

defaultQueueSize

Condition: (1) Ignored on AIX

```
static long defaultQueueSize();
```

Obtains the default message queue size to use for new IThreads.

Motif Information: You cannot adjust the queue size on the X environment. This function returns the queue size that is set using IThread::setDefaultQueueSize. Initially, this value is 0.

Presentation Manager Information: Obtains the default Presentation Manager message queue size to use for new IThreads.

defaultStackSize

Condition: (1) Ignored on AIX

```
static unsigned long defaultStackSize();
```

Obtains the default stack size. The library uses this value by default for new IThreads.

Motif Information: 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. Initially, this value is 32768.

id

```
virtual IThreadId id() const;
```

Obtains the ID of the thread. 0 indicates the thread is not started.

isStarted

```
virtual Boolean isStarted() const;
```

If the thread is currently active, true is returned.

messageQueue

Condition: (2) OS/2 Only

```
IMessageQueueHandle messageQueue() const;
```

Condition: (2) OS/2 Only

```
IMessageQueueHandle messageQueue();
```

Returns the handle of the message queue.

priorityClass

Condition: (1) Ignored on AIX

```
virtual IApplication::PriorityClass priorityClass() const;
```

Obtains the priority class of the thread. The returned value is an enumerator provided by IApplication::PriorityClass (see page 30).

Motif Information: The priority class for AIX is always IApplication::regular.

priorityLevel

Condition: (1) Ignored on AIX

```
virtual unsigned priorityLevel() const;
```

Obtains the priority level of the thread.

Motif Information: The level for AIX is always 0.

Presentation Manager Information The value of the level is in the range 0-31.

queueSize

Condition: (1) Ignored on AIX

```
virtual long queueSize() const;
```

Obtains the message queue size for the thread. This is the thread used by the application to receive events from the operating system.

Motif Information: You cannot control the size of the message queue in the X environment. This function returns the value set by IThread::setQueueSize. The default value is defaultQueueSize.

relatedHandlesList

Condition: (3) AIX only

```
IRelatedHandlesList *relatedHandlesList() const;
```

Obtains a pointer for the related handles list for this thread.

resume

Condition: (1) Ignored on AIX

```
virtual void resume();
```

Resumes the thread's execution.

Motif Information: The AIX release of the User Interface Class Library supports only a single-threaded environment. Therefore, you cannot suspend or resume a thread.

setAutoInitGUI

```
virtual IThread &setAutoInitGUI(Boolean initFlag = true);
```

Sets the flag that automatically initializes the GUI for this thread.

Motif Information: 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.

setAutoInitPM

```
virtual IThread &setAutoInitPM(Boolean initFlag = true);
```

Sets the flag that automatically initializes OS/2 Presentation Manager for this thread.

Note: This function is obsolete. Use IThread::setAutoInitGUI instead.

setDefaultAutoInitGUI

```
static void setDefaultAutoInitGUI(Boolean initFlag = true);
```

Sets the default flag that automatically initializes the GUI.

Motif Information: 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.

setDefaultAutoInitPM

```
static void setDefaultAutoInitPM(Boolean initFlag = true);
```

Sets the default flag that automatically initializes OS/2 Presentation Manager.

Note: This function is obsolete. Use IThread::setDefaultAutoInitGUI instead.

setDefaultQueueSize

Condition: (1) Ignored on AIX

```
static void setDefaultQueueSize(long queueSize);
```

Sets the default message queue size for the GUI.

Motif Information: 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.

setDefaultStackSize

Condition: (1) Ignored on AIX

```
static void setDefaultStackSize(unsigned long aSize);
```

Sets the default stack size.

Motif Information: 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.

setPriority

Condition: (1) Ignored on AIX

```
virtual IThread &setPriority(
                    IApplication::PriorityClass aClass,
                    unsigned level);
```

Sets the priority class and level of the thread.

Motif Information: You cannot control the priority of a thread in the AIX environment. This function has no effect on the priority of an executing thread.

setQueueSize

Condition: (1) Ignored on AIX

```
virtual IThread &setQueueSize(long queueSize);
```

Sets the GUI message queue size for this thread.

Motif Information: 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.

setRelatedHandlesList

Condition: (3) AIX only

```
IThread &setRelatedHandlesList(IRelatedHandlesList *list);
```

Sets a pointer to the related handles list for this thread.

setStackSize

Condition: (1) Ignored on AIX

```
virtual IThread &setStackSize(unsigned long aSize);
```

Sets the thread stack size, in bytes. If the thread is started, this value takes effect only when the thread is stopped and restarted.

Motif Information: 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.

setVariable

```
IThread &setVariable(
            const IString &key,
            const IString &value);
```

Lets users store data per thread that can later be retrieved by IThread::variable (see page 899).

setWindowList

```
IThread &setWindowList(IWindowList *list);
```

Sets a pointer to the window information list for this thread.

stackSize

Condition: (1) Ignored on AIX

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

Obtains the thread stack size, in bytes.

Motif Information: Returns the value of the stack size that was set with the function setStackSize. You cannot control the size of the stack in AIX; therefore, this is not really the size of the stack. The default value is defaultStackSize.

start

Condition: (2) OS/2 Only

```
void start(
       const IReference < IThreadFn > &aFnObjRef,
       Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

Condition: (2) OS/2 Only

```
void start(
       OptlinkFnPtr pfn,
       void *anArg,
       Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

Condition: (2) OS/2 Only

```
void start(
       SystemFnPtr pfn,
       unsigned long anArg,
       Boolean autoInitGUI = IThread::defaultAutoInitGUI ( ));
```

Starts an asynchronous thread. You can start threads with an arbitrary IThreadFn (see page 902). The constructor passes the IThreadFn using an IReference (see page 727) so that the IThreadFn can be deleted (if necessary) when the thread terminates.

Motif Information: Because the User Interface Class Library supports only single threading there is no need to call this function.

Presentation Manager Information 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 startable using _beginthread.
- With an IThread::SystemFnPtr type and unsigned long parameter. This is to provide support for functions otherwise startable using DosCreateThread.

stop

```
virtual void stop();
```

Stops the thread.

stopProcessingMsgs

```
virtual IThread &stopProcessingMsgs();
```

Terminates the processing of window events (messages) initiated by ICurrentThread::processMsgs (see page 236).

suspend

Condition: (1) Ignored on AIX

```
virtual void suspend();
```

Suspends the thread's execution.

Motif Information: The User Interface Class Library supports only single threading, therefore suspend and resume have no effect.

variable
```
IString variable(const IString &key) const;
```

Returns the data associated with the key that was stored using IThread::setVariable (see page 897).

windowList
```
IWindowList *windowList() const;
```

Obtains a pointer to the window information list for this thread.

Figure 310. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

newStartedThread
```
static IStartedThread *newStartedThread();
```

Returns a pointer to the new IStartedThread.

operator =
```
IThread &operator =(const IThread &aThread);
```

Provides for the assignment of one IThread to another, therefore preserving resource allocation.

startedThread
```
virtual IStartedThread *startedThread() const;
```

Returns the address of the corresponding IStartedThread.

Chapter 209. IThread::Cursor

Derivation

```
IBase
  IVBase
    IThread::Cursor
```

Header File: ithread.hpp

The nested class IThread::Cursor defines objects that you can use to traverse or iterate over 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:

1. Calling setToFirst (see page 901)
2. Looping through the threads using setToNext (see page 901)
3. Processing the returned thread IDs from threadId (see page 901) until isValid (see page 900) returns false

Note: This class only provides access to threads that IThread (see page 890) is aware of.

Constructors

```
public:
Cursor(Boolean allThreads = true);
```

The constructor accepts an optional flag indicating whether all threads are enumerated or only those that have User Interface Class Library windows. The default is all threads.

Public Members

invalidate

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

Invalidates the cursor.

isValid

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

If the cursor is valid, true is returned.

© Copyright IBM Corp. 1994

setToFirst

```
Boolean setToFirst();
```

Initializes the cursor to the first thread.

setToNext

```
Boolean setToNext();
```

Advances the cursor to the next thread.

threadId

```
IThreadId threadId() const;
```

Returns the ID of the thread at the current cursor position.

Figure 311. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 210. IThreadFn

Derivation

IBase
  IVBase
    IRefCounted
      IThreadFn

Header File: ithread.hpp

Objects of the IThreadFn class represent functions to be dispatched on secondary threads of execution when an IThread (see page 890) is started. The library reference counts these objects to manage their destruction after the thread has terminated. The library passes such a reference to IThread::start (see page 897).

This class is an abstract thread function class.

Constructors

```
public:
IThreadFn();
```

You cannot construct objects of this class, because it is an abstract class.

Public Members

run

```
virtual void run() = 0;
```

Called when the thread function object is called.

Figure 312. Inherited Public Members

Member function	Class
addRef	IRefCounted
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*
removeRef	IRefCounted
useCount	IRefCounted

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 211. IThreadId

Derivation

IBase
  IHandle
    IThreadId

Header File: ihandle.hpp

Objects of the IThreadId class access numeric identifiers for threads.

Motif Information: In AIX, the User Interface Class Library does not support multiple threads. This class always returns a ThreadId of 1.

Presentation Manager Information: IThreadId is an alias for the OS/2 Programmer's Toolkit type TID.

Constructors

```
public:
IThreadId(Value tid = 0);
```

You can construct objects of this class from a thread ID (a value of type IHandle::Value), which defaults to 0.

Public Members

operator unsigned long

```
operator unsigned long() const;
```

Returns the thread ID as an unsigned long.

operator Value

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

Returns the IThreadId value.

Figure 313. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle

Figure 313. Inherited Public Members

Member function	Class
operator <<	IBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 212. IThreadMemberFn

Derivation

```
IBase
  IVBase
    IRefCounted
      IThreadFn
        IThreadMemberFn
```

Header File: ithread.hpp

The IThreadMemberFn template class is derived from IThreadFn (see page 902) for dispatching C++ member functions against an object on a new thread. The template argument is the class of the object the functions are called for.

The constructor for such objects requires:

- A reference to the object the member functions are to be applied against
- A pointer to the member functions

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 instantiated with the following template argument:

T The class of object against which the member functions are applied.

Constructors

```
public:
IThreadMemberFn(T &obj, void ( T::* mem ) ( ));
```

The constructor requires the following:

- An object of the template argument class T
- A pointer to a member function of that class, returning void and accepting no parameters

Public Members

run

```
virtual void run();
```

Calls the pointed-to member function to the argument provided at construction.

© Copyright IBM Corp. 1994

Figure 314. Inherited Public Members

Member function	Class
addRef	IRefCounted
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*
removeRef	IRefCounted
run	IThreadFn
useCount	IRefCounted

Note: * Application Support Class Library

Chapter 213. ITitle

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            ITitle
```

Header File: ititle.hpp

Objects of the ITitle class create and update 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 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.

Presentation Manager Information: Presentation Manager does not allow the length of any title to exceed 60 characters. If you try to set the title text to a length greater than 60, the text is truncated.

Constructors

```
public:
ITitle(
   IWindow *owner,
   const char *objectName = 0,
   const char *viewName = 0,
   unsigned long viewNumber = 0);

public:
ITitle(
   IWindow *owner,
   const IResourceId &objectName,
   const IResourceId &viewName = 0,
   unsigned long viewNumber = 0);
```


© Copyright IBM Corp. 1994

You can construct objects of this class from the following:

- char*
- IResourceId of STRINGTABLE items

Public Members

activeColor

Condition: (1) Ignored on AIX

```
virtual IColor activeColor() const;
```

Returns the active color value of the title area. If you have not set the color for the area, the default is returned.

activeTextBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IColor activeTextBackgroundColor() const;
```

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.

activeTextForegroundColor

Condition: (1) Ignored on AIX

```
virtual IColor activeTextForegroundColor() const;
```

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.

borderColor

Condition: (1) Ignored on AIX

```
virtual IColor borderColor() const;
```

Returns the border color value of the title area. If you have not set the color for the area, the default is returned.

enableNotification

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

Causes the title to send notifications to any added observer objects.

inactiveColor
Condition: (1) Ignored on AIX

```
virtual IColor inactiveColor() const;
```

Returns the inactive color value of the title area. If you have not set the color for the area, the default is returned.

inactiveTextBackgroundColor
Condition: (1) Ignored on AIX

```
virtual IColor inactiveTextBackgroundColor() const;
```

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.

inactiveTextForegroundColor
Condition: (1) Ignored on AIX

```
virtual IColor inactiveTextForegroundColor() const;
```

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.

objectText
```
virtual IString objectText() const;
```

Returns the object text.

resetActiveTextBackgroundColor
Condition: (1) Ignored on AIX

```
virtual ITitle &resetActiveTextBackgroundColor();
```

Resets the active text background color by undoing a previous set.

resetActiveTextForegroundColor
Condition: (1) Ignored on AIX

```
virtual ITitle &resetActiveTextForegroundColor();
```

Resets the active text foreground color by undoing a previous set.

resetInactiveTextBackgroundColor
Condition: (1) Ignored on AIX

```
virtual ITitle &resetInactiveTextBackgroundColor();
```

Resets the inactive text background color by undoing a previous set.

resetInactiveTextForegroundColor

Condition: (1) Ignored on AIX

```
virtual ITitle &resetInactiveTextForegroundColor();
```

Resets the inactive text foreground color by undoing a previous set.

setActiveTextBackgroundColor

Condition: (1) Ignored on AIX

```
virtual ITitle &setActiveTextBackgroundColor(
                  const IColor &color);
```

Sets the active text background color to the specified color. The window area is identified by a system-defined presentation parameter value.

setActiveTextForegroundColor

Condition: (1) Ignored on AIX

```
virtual ITitle &setActiveTextForegroundColor(
                  const IColor &color);
```

Sets the active text foreground color to the specified color. The window area is identified by a system-defined presentation parameter value.

setInactiveTextBackgroundColor

Condition: (1) Ignored on AIX

```
virtual ITitle &setInactiveTextBackgroundColor(
                  const IColor &color);
```

Sets the inactive text background color to the specified color. The window area is identified by a system-defined presentation parameter value.

setInactiveTextForegroundColor

Condition: (1) Ignored on AIX

```
virtual ITitle &setInactiveTextForegroundColor(
                  const IColor &color);
```

Sets the inactive text foreground color to the specified color. The window area is identified by a system-defined presentation parameter value.

setObjectText

```
virtual ITitle &setObjectText(
                  const IResourceId &objectNameResId);

virtual ITitle &setObjectText(const char *objectName);
```

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 (see page 875) instead of this function.

setText

Condition: (3) AIX only

```
virtual ITitle &setText(const IResourceId &textResId);
```

Condition: (3) AIX only

```
virtual ITitle &setText(const char *text);
```

Sets the title text.

setTitleText

```
virtual ITitle &setTitleText(
                  const char *objectName,
                  const char *viewName = 0,
                  unsigned long viewNum = 0);

virtual ITitle &setTitleText(
                  const IResourceId &objectNameResId,
                  const IResourceId &viewNameResId = 0,
                  unsigned long viewNum = 0);
```

Sets all three components of a title (object text, view text, and view number) at once.

setViewNum

```
virtual ITitle &setViewNum(unsigned long viewNum);
```

Sets the view number.

setViewText

```
virtual ITitle &setViewText(
                  const IResourceId &viewNameResId);

virtual ITitle &setViewText(const char *viewName);
```

Sets the view text.

text

Condition: (3) AIX only

```
virtual IString text() const;
```

Returns the title text.

textLength

Condition: (3) AIX only

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

Returns the current length of the title text, in bytes.

viewNum

```
unsigned long viewNum() const;
```

Returns the view number.

viewText

```
virtual IString viewText() const;
```

Returns the view text.

Figure 315. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
displaySize	ITextControl
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow

Figure 315. Inherited Public Members

Member function	Class
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow

Figure 315. Inherited Public Members

Member function	Class
isTabStop	IControl
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow

Figure 315. Inherited Public Members

Member function	Class
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ITextControl
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
setText	ITextControl
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
text	ITextControl
textLength	ITextControl
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

Figure 316. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
calcMinimumSize	IWindow
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
prepareForUse	IWindow
registerCallbacks	IWindow
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 214. IViewPort

Derivation

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ICanvas
            IViewPort

Header File: ivport.hpp

Objects of the IViewPort class provide scrolling support for the view window. Typically, the view window does not readily support resizing. Because of this, you must support scrolling so users can scroll the view window to see clipped information.

View window candidates include the following classes:

ICanvas (see page 63)
ISetCanvas (see page 768)
INotebook (see page 579)
IContainerControl (see page 149)

You specify the view window by making it a child window of the view port. A view port only supports one view window. The view port changes the parent window of the view window to a scrollable rectangular window to properly clip the child window.

The IViewPort object optionally includes horizontal and vertical scroll bars. You can specify scroll bars that are the following:

- Always visible
- Never visible
- Visible only if the size of the view window is larger than the size of the view port window

If the view port has nothing to scroll because the view window is entirely visible, the scroll bars are hidden.

Logical sizes specified through IViewPort::setViewWindowSize are limited to 0x3FFF0000. If you do not call setViewWindowSize, the size of a view window is limited to 32 K. If the view window has no size and does not use a logical size, IViewPort sizes the view window to its minimum size.

Note: Only one window can be a child of the IViewPort window. If you try to give the view port control more than one child, an IInvalidRequest exception is thrown.

Portability Considerations: The AIX release of the User Interface Class Library does not support the style combination of one scroll bar displayed as needed and the other scroll bar always displayed. If this combination of styles is used, both scroll bars are

© Copyright IBM Corp. 1994

always displayed. Motif supports all other combinations for displaying the view port scroll bars.

Note: Previous versions of this class created each scroll bar with a 1-based range. The scroll bar ranges are now 0-based so that the interface is consistent with scrollViewHorizontallyTo (see page 916) and scrollViewVerticallyTo (see page 916).

Nested Classes

IViewPort::Style

Constructors

```
public:
IViewPort(
   unsigned long windowId,
   IWindow *parent,
   IWindow *owner,
   const IRectangle &initial = IRectangle ( ),
   const Style &style = defaultStyle ( ));
```

You can construct objects of this class using the following parameters:

windowId Window identifier of the canvas you construct.

Note: For OS/2, this value should be limited to the range 0 through 65535.

It is recommended that you:

- Give unique identifiers to all windows in the same frame window. *In the same window* means that the frame window is the first frame in their parent window chain.
- Give the client window a window ID of FID_CLIENT (0x8008).
- Do not use the other FID_xxx values.

The OS/2 version of IViewPort class reserves the following window IDs for child windows that it creates:

IC_VIEWPORT_VERTSCROLLBAR
Scroll bar.

IC_VIEWPORT_HORZSCROLLBAR
Scroll bar.

IC_VIEWPORT_VIEWRECTANGLE
Scroll rectangle. The window within which the application window is scrolled. IViewPort changes the parent of the view window to this window.

parent The parent window of the canvas you construct. You must specify a parent window. This constructor throws an IInvalidParameter exception if you pass an IWindow* of 0.

The parent window is primarily used for visible relationships.

owner Owner window of the canvas you construct.

Note: In OS/2, the owner window is primarily used for routing notification events and unprocessed messages. OS/2 also uses the owner window chain to inherit colors.

In Motif, the owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial (Optional) The initial position and size of the canvas you construct. The initial position is the lower-left corner of the canvas relative to the lower-left corner of the parent window.

style (Optional) The window's characteristics. These can be a combination of the IViewPort::Style and the IWindow::Style.

Public Members

defaultStyle

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

Returns the default style. The default style is classDefaultStyle (see page 922) unless you change the style using setDefaultStyle (see page 917).

horizontalScrollBar

```
IScrollBar *horizontalScrollBar() const;
```

Returns the address of the horizontal scroll bar.

scrollViewHorizontallyTo

```
virtual IViewPort &scrollViewHorizontallyTo(
                    unsigned long leftOffset);
```

Scrolls the view window horizontally.

leftOffset The 0-based offset from the left edge of the view window to scroll the left edge of the view port to.

scrollViewVerticallyTo

```
virtual IViewPort &scrollViewVerticallyTo(
                    unsigned long topOffset);
```

Scrolls the view window vertically.

topOffset The 0-based offset from the top edge of the view window to scroll the top edge of the view port.

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

Sets the default style for all subsequent view ports.

style Use the styles provided by IViewPort::Styles (see page 922) to specify the default style.

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

Processes a size change as a layout change.

setViewWindowSize
```
virtual IViewPort &setViewWindowSize(const ISize &size);
```

Changes the logical size of the window being scrolled. The logical size does not have to match the real size of the window.

If you call this function, you will manage the appearance of the view window. IViewPort processes the position and size of the scroll boxes of its scroll bars, but you are responsible for changing the position or repainting the view window to give the appearance of scrolling.

size New logical size of the view window.

verticalScrollBar
```
IScrollBar *verticalScrollBar() const;
```

Returns the address of the vertical scroll bar.

viewWindow
```
virtual IWindowHandle viewWindow();
```

Returns the handle of the window bounded by the view port. If more than one child of the view port is found, an exception is thrown. The parent of the view window is changed to the scrollable rectangular window so that the view window is properly clipped.

viewWindowDrawRectangle
```
virtual IRectangle viewWindowDrawRectangle() const;
```

Returns the area of the view window that is currently visible in the view port.

viewWindowSize

```
virtual ISize viewWindowSize() const;
```

Returns the logical size of the window being scrolled. If you have not called IViewPort::setViewWindowSize (see page 917) to set a logical size, the real size of the view window is returned.

Figure 317. Inherited Public Members

Member function	Class
activeColor	IWindow
asDebugInfo	IBase*
asDebugInfo	IVBase*
asDebugInfo	IWindow
asString	IBase*
asString	IVBase*
asString	IWindow
backgroundColor	IWindow
backgroundColor	ICanvas
borderColor	IWindow
characterSize	IWindow
childAt	IWindow
defaultPushButton	IWindow
defaultPushButton	ICanvas
disable	IWindow
disabledBackgroundColor	IWindow
disabledForegroundColor	IWindow
disableGroup	IControl
disableNotification	INotifier
disableNotification	IWindow
disableTabStop	IControl
disableUpdate	IWindow
enable	IWindow
enableGroup	IControl
enableNotification	INotifier
enableNotification	IWindow
enableTabStop	IControl
enableUpdate	IWindow
font	IWindow
foregroundColor	IWindow
handle	IWindow
handleException	IWindow
hasFocus	IWindow
hide	IWindow

Figure 317. Inherited Public Members

Member function	Class
hideSourceEmphasis	IWindow
hiliteBackgroundColor	IWindow
hiliteForegroundColor	IWindow
id	IWindow
inactiveColor	IWindow
isAutoDeleteObject	IWindow
isAutoDestroyWindow	IWindow
isDisabled	IWindow
isEnabled	IWindow
isEnabledForNotification	INotifier
isEnabledForNotification	IWindow
isFrameWindow	IWindow
isGroup	IWindow
isGroup	IControl
isLayoutDistorted	IWindow
isShowing	IWindow
isTabStop	IWindow
isTabStop	IControl
isTabStop	ICanvas
isValid	IWindow
isVisible	IWindow
itemProvider	IWindow
layoutAdjustment	IWindow
matchForMnemonic	IWindow
matchForMnemonic	ICanvas
messageQueue	IWindow
minimumSize	IWindow
moveSizeTo	IWindow
moveTo	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
operator <<	IBase*
operator <<	IVBase*
origDefaultButtonHandle	ICanvas

Figure 317. Inherited Public Members

Member function	Class
owner	IWindow
parent	IWindow
position	IWindow
positionBehindSibling	IWindow
positionBehindSiblings	IWindow
positionOnSiblings	IWindow
postEvent	IWindow
presSpace	IWindow
rect	IWindow
refresh	IWindow
releasePresSpace	IWindow
resetActiveColor	IWindow
resetBackgroundColor	IWindow
resetBorderColor	IWindow
resetDisabledBackgroundColor	IWindow
resetDisabledForegroundColor	IWindow
resetForegroundColor	IWindow
resetHiliteBackgroundColor	IWindow
resetHiliteForegroundColor	IWindow
resetInactiveColor	IWindow
resetMinimumSize	IWindow
resetShadowColor	IWindow
sendEvent	IWindow
setActiveColor	IWindow
setAutoDeleteObject	IWindow
setAutoDestroyWindow	IWindow
setBackgroundColor	IWindow
setBorderColor	IWindow
setDisabledBackgroundColor	IWindow
setDisabledForegroundColor	IWindow
setFocus	IWindow
setFont	IWindow
setForegroundColor	IWindow
setHiliteBackgroundColor	IWindow
setHiliteForegroundColor	IWindow
setId	IWindow
setInactiveColor	IWindow
setItemProvider	IWindow
setLayoutDistorted	IWindow
setLayoutDistorted	ICanvas

Figure 317. Inherited Public Members

Member function	Class
setMinimumSize	IWindow
setOwner	IWindow
setParent	IWindow
setShadowColor	IWindow
shadowColor	IWindow
show	IWindow
showSourceEmphasis	IWindow
size	IWindow
sizeTo	IWindow
topHandle	IWindow
visibleRectangle	IWindow

Note: * Application Support Class Library

Protected Members

calcMinimumSize

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

Returns the recommended minimum size for this IViewPort object. The size is based on the minimum size of the vertical and horizontal scroll bars.

layout

```
virtual IViewPort &layout();
```

Positions the scroll bars, scroll rectangle window, and view window.

positionViewWindow

```
virtual IViewPort &positionViewWindow(
                    const IWindowHandle &viewWindow,
                    const IRectangle &viewRectangle);
```

Positions the view window within the view rectangle.

viewWindow
: Window handle of the view window.

viewRectangle
: Rectangle whose size defines how much of the view window can display in the view window. You can query the top-left corner of the view window that is visible in the rectangle using IViewPort::topLeftViewPoint. positionViewWindow is called by the IViewPort::scrollViewHorizontallyTo, scrollViewVerticallyTo, and layout member functions.

setTopLeftViewPoint

```
virtual IViewPort &setTopLeftViewPoint(
                    const IPoint &topLeft);
```

Sets the view window coordinate that displays in the upper-left corner of the view rectangle.

topLeft
: The coordinate in the view window that the IViewPort considers visible in the upper-left corner of the view port. To scroll the view window, call IViewPort::scrollViewHorizontallyTo or scrollViewVerticallyTo instead of setTopLeftViewPoint. This function does not update the scroll bars.

setupScrollBars

```
virtual IViewPort &setupScrollBars();
```

Sizes, shows, or hides scroll bars for the view port window, as appropriate. This function also positions and sizes the scroll rectangle window that bounds the view window. IViewPort calls this function before calling positionViewWindow.

topLeftViewPoint

```
IPoint topLeftViewPoint() const;
```

Returns the view window coordinate that displays in the upper-left corner of the view rectangle.

Figure 318. Inherited Protected Members

Member function	Class
addHandler	IWindow
addObserver	INotifier
addObserver	IWindow
areChildrenReversed	ICanvas
calcMinimumSize	IWindow
calcMinimumSize	ICanvas
create	IWindow
defaultProcedure	IWindow
deleteIsInProcess	IWindow
dispatch	IWindow
fixupChildren	ICanvas
isPrimaryWindow	IWindow
isRelatedHandle	IWindow
layout	ICanvas
layoutSize	ICanvas
notificationHandler	IWindow
notifyObservers	INotifier
notifyObservers	IWindow
observerList	INotifier
observerList	IWindow
passEventToOwner	IWindow
passEventToOwner	ICanvas
prepareForUse	IWindow
registerCallbacks	IWindow
registerCallbacks	ICanvas
removeAllObservers	INotifier
removeAllObservers	IWindow
removeHandler	IWindow
removeObserver	INotifier
removeObserver	IWindow
resetColor	IWindow
setLayoutSize	ICanvas
setNotificationHandler	IWindow
setStyle	IWindow
setWindowData	IWindow

Figure 318. Inherited Protected Members

Member function	Class
startHandlingEventsFor	IWindow
style	IWindow
unregisterCallbacks	IWindow
unregisterCallbacks	ICanvas
windowULong	IWindow
windowUShort	IWindow

Note: * Application Support Class Library

Chapter 215. IViewPort::Style

Derivation

IBase
  IBitFlag
    IViewPort::Style

Header File: ivport.hpp

The nested class IViewPort::Style provides a valid set of view port styles for the defaultStyle and setDefaultStyle functions, and for the constructor of the IViewPort (see page 914). You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (see page 70)
IWindow::Style (see page 957)

Styles

alwaysHorizontalScrollBar
Specifies that the view port never removes the horizontal scroll bar.

alwaysVerticalScrollBar
Specifies that the view port never removes the vertical scroll bar.

asNeededHorizontalScrollBar
Specifies that the view port removes the horizontal scroll bar when the entire width of the view window is visible and there is no data to scroll horizontally.

asNeededVerticalScrollBar
Specifies that the view port removes the vertical scroll bar when the entire height of the view window is visible and there is no data to scroll vertically.

classDefaultStyle
Specifies the original default style for this class, which is asNeededHorizontalScrollBar | asNeededVerticalScrollBar | IWindow::visible.

noHorizontalScrollBar
Specifies that the view port never shows the horizontal scroll bar.

noVerticalScrollBar
Specifies that the view port never shows the vertical scroll bar.

noViewWindowFill
Specifies that the view port does not paint the portion of the view rectangle occupied by the view window.

Note: If you know that the view window paints its entire rectangle, specify this style to optimize overall painting.

© Copyright IBM Corp. 1994

Portability Considerations: The AIX release of the User Interface Class Library does not support having one scroll bar created with the style asNeeded and the other scroll bar with the style always. If you use this combination of styles, both scroll bars are always displayed. All other combinations of displaying scroll bars are supported.

Public Members

Figure 319. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 320. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 216. IWindow

Derivation

```
IBase
  IVBase
    INotifier
      IWindow
```

Header File: iwindow.hpp

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.

Portability Considerations: This class is portable across OS/2 and AIX. Most of the member functions are also portable. See individual member functions for details.

Motif Information: In Motif, the User Interface Class Library does not support dialog templates. However, there is still an IWindow constructor which 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.

Nested Classes

IWindow::ExceptionFn
IWindow::Style
IWindow::ChildCursor

Constructors

```
public:
IWindow(unsigned long id, IWindow *parentDialog);

public:
IWindow(const IWindowHandle &handle);
```


© Copyright IBM Corp. 1994

Condition: (4) Typically for internal use

```
protected:
IWindow();
```

The User Interface Class Library provides two constructors so you can create a user interface “wrapper” object for a window of unknown type, such as any window in the client area. The third constructor, the default, is only used internally by the library.

You can construct an IWindow from:

- A control on a loaded dialog template
- An existing window’s window handle

The User Interface Class Library implements both constructors by calling:

- IWindow::setAutoDestroyWindow(false) (see page 938) ensures the library does not destroy the window when the IWindow is destructed.
- IWindow::startHandlingEventsFor (see page 951), which subclasses and adds the window to the window list for full event processing.

Public Members

activeColor

Condition: (1) Ignored on AIX

```
virtual IColor activeColor() const;
```

Returns the active color value of the window area. If you have not set the color for the area, the default color is returned.

asDebugInfo

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

Returns a string containing detailed information about the window, including the information provided by IWindow::asString (see page 925).

asString

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

Returns a string containing the window handle and ID.

backgroundColor

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

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

borderColor
```
virtual IColor borderColor() const;
```

Returns the border color value of the window area. If you have not set the color for the area, the default color is returned.

characterSize
```
ISize characterSize() const;
```

Calculates and returns the average character width and maximum character height for the currently set font.

childAt
```
IWindowHandle childAt(const ChildCursor &aCursor) const;
```

Returns the window handle of the child window at the current position of the specified cursor.

defaultOrdering
```
static SiblingOrder defaultOrdering();
```

Returns the order in which the library creates the new windows relative to their sibling windows.

defaultPushButton
```
virtual IWindowHandle defaultPushButton() const;
```

Returns the first child window that is a default push button, if one exists.

desktopWindow
```
static IWindow *desktopWindow();
```

Returns the object representing the system desktop window.

Motif Information: 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.

disable
```
virtual IWindow &disable();
```

Prevents keyboard and mouse input from being sent to the window.

disabledBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IColor disabledBackgroundColor() const;
```

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.

disabledForegroundColor

Condition: (1) Ignored on AIX

```
virtual IColor disabledForegroundColor() const;
```

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.

disableNotification

```
virtual IWindow &disableNotification();
```

Causes the window to stop sending notifications to all added observer objects.

disableUpdate

```
virtual IWindow &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 (see page 942) 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.

Motif Information: Motif ignores this function, as the X Window System is designed to avoid the need to disable or enable during most complex drawing.

enable

```
virtual IWindow &enable(Boolean enableWindow = true);
```

Enables the window to accept keyboard and mouse input.

enableNotification

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

Causes the window to send notifications to any added observer objects.

enableUpdate

```
virtual IWindow &enableUpdate(Boolean enableWindow = true);
```

Draws changes to a window on the screen.

Motif Information: Motif ignores this function, as the X Window System is designed to avoid the need to disable or enable during most complex drawing.

exceptionFunction

```
static IWindow::ExceptionFn *exceptionFunction();
```

Returns a pointer to the current exception function. If one has not been registered, 0 is returned.

font

```
virtual IFont font() const;
```

Returns the font used by the window.

foregroundColor

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

Returns the foreground color value of the window area, or the default if no color for the area has been set.

handle

```
virtual IWindowHandle handle() const;
```

Returns the window handle.

handleException

```
virtual Boolean handleException(
                IException &dispatcherException,
                IEvent &exceptionEvent);
```

Called when the library detects an exception during the dispatch of an event. The default behavior determines if an IWindow::ExceptionFn (see page 956) object has been registered and calls it. Otherwise, false is returned and the exception is thrown again.

handleWithParent

```
static IWindowHandle handleWithParent(
                     unsigned long id,
                     const IWindowHandle &parent);
```

Returns the window handle for the specified ID and parent.

Note: This function replaces IWindow::handleWithParent.

Motif Information: 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.

hasFocus

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

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

hide

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

Hides the window.

hideSourceEmphasis

```
virtual IWindow &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.

hiliteBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IColor hiliteBackgroundColor() const;
```

Returns the highlight background color value of the window area, or the default if no color for the area has been set.

hiliteForegroundColor

Condition: (1) Ignored on AIX

```
virtual IColor hiliteForegroundColor() const;
```

Returns the highlight foreground color value of the window area, or the default if no color for the area has been set.

id

```
unsigned long id() const;
```

Returns the window ID.

Motif Information: 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 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 which 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.

inactiveColor

Condition: (1) Ignored on AIX

```
virtual IColor inactiveColor() const;
```

Returns the inactive color value of the window area, or the default if no color for the area has been set.

isAutoDeleteObject

```
Boolean isAutoDeleteObject() const;
```

Returns true if the window object is deleted when a destroy event is dispatched to the window.

isAutoDestroyWindow

```
Boolean isAutoDestroyWindow() const;
```

Returns true if the presentation window is destroyed when the window object is deleted.

isDisabled

```
Boolean isDisabled() const;
```

If the window is not sent mouse and keyboard input, true is returned.

Note: This function is obsolete. Use IWindow::isEnabled instead.

isEnabled

```
Boolean isEnabled() const;
```

If the window is sent mouse and keyboard input, true is returned.

isEnabledForNotification

```
virtual Boolean isEnabledForNotification() const;
```

Returns true if the window is sending notifications to observer objects.

isFrameWindow

```
virtual Boolean isFrameWindow() const;
```

If this object represents a frame window, true is returned.

isGroup

```
virtual Boolean isGroup() const;
```

If the control has the group style set, true is returned.

isLayoutDistorted

```
virtual Boolean isLayoutDistorted(
                unsigned long layoutAttribute) const;
```

Determines whether changes have been made in a window that require updating the layout of the window in a canvas.

isShowing

```
Boolean isShowing() const;
```

If any part of the window is showing, true is returned.

Note: A window can be visible and yet not be showing if it is covered by another window.

isTabStop

```
virtual Boolean isTabStop() const;
```

If the user can tab to the window, true is returned.

isValid

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

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.

isVisible

```
Boolean isVisible() const;
```

If the window's style is set to visible, true is returned.

Note: A window can have the style visible (see page 957) and yet not be showing if it is covered by another window.

itemProvider

```
IDMItemProvider *itemProvider() const;
```

Returns a pointer to the drag item provider (IDMItemProvider) for this window.

Motif Information: AIX does not support this function.

layoutAdjustment

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

The layout adjustment is the dimensions that a window is moved and/or sized after a canvas runs its layout routines. This function exists to support controls like the drop-down combination box where the displayed size of the control is different than its actual size. The default behavior of this function returns IRectangle(0,0,0,0).

mapPoint

```
static IPoint mapPoint(
               const IPoint &aPoint,
               const IWindowHandle &from,
               const IWindowHandle &to);
```

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.

matchForMnemonic

```
virtual IWindowHandle matchForMnemonic(
                       unsigned short character) const;
```

Returns the first child window using the specified character as a mnemonic.

messageQueue

Condition: (2) OS/2 Only

```
IMessageQueueHandle messageQueue() const;
```

Returns the handle for the window's message queue.

minimumSize

```
ISize minimumSize(
        Boolean windowCalculatedSize = false) const;
```

Returns the minimum allowable size set by a derived class. If setMinimumSize has not been called or windowCalculatedSize is true, it returns the value of calcMinimumSize.

moveSizeTo

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

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. In this release, your application must use Presentation Manager coordinates.

moveTo

```
virtual IWindow &moveTo(const IPoint &aPoint);
```

Changes the position of the window, converting from the coordinates understood by this application to window system coordinates, if they are different.

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. In this release, your application must use Presentation Manager coordinates.

notifyObservers

```
virtual IWindow &notifyObservers(
                    const INotificationEvent &anEvent);
protected:
virtual IWindow &notifyObservers(
                    const INotificationId &nId);
```

Notifies all observers in part's collection.

objectWindow

```
static IWindow *objectWindow();
```

Returns the instance of the object representing the system object window.

Motif Information: The object window is the same as the desktop window when on X/Motif systems. In X, both widgets destined to be visible and purely logical, non-visual 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.

owner

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

Returns the window's owner. This function may return 0 if:

- The window was created without an owner
- The owner window is not represented by an IWindow object

Since a pointer value of 0 can cause unpredictable behavior, check the value of the returned pointer before using it.

Motif Information: 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 which do not themselves support it.

parent

```
IWindow *parent() const;
```

Returns the window's parent. This function may return 0 if:

- The window was created without an parent
- The parent window is not represented by an IWindow object

Since a pointer value of 0 can cause unpredictable behavior, you should check the value of the returned pointer before using it.

Motif Information: 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 either:

- An IWindow
- The desktopWindow
- The objectWindow

Use this function instead of XtParent wherever possible.

position

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

Converts the window's position from window system coordinates to coordinates understood by your application (if they are different).

Note: In this release, your application must use OS/2 Presentation Manager coordinates.

positionBehindSibling

```
virtual IWindow &positionBehindSibling(
                  const IWindowHandle &siblingWindow);
```

Puts this window in the z-order behind the specified sibling window.

positionBehindSiblings

```
virtual IWindow &positionBehindSiblings();
```

Puts this window in the z-order behind all its sibling windows.

positionOnSiblings

```
virtual IWindow &positionOnSiblings();
```

Puts this window in the z-order on top of all its sibling windows.

postEvent

```
const IWindow &postEvent(
                  unsigned long eventId,
                  const IEventParameter1 &parm1 = 0,
                  const IEventParameter2 &parm2 = 0) const;

const IWindow &postEvent(const IEvent &event) const;

const IWindow &postEvent(
                  EventType eventType,
                  const IEventParameter1 &parm1 = 0,
                  const IEventParameter2 &parm2 = 0) const;
```

Posts an event constructed from the arguments to the window. If you specify an enumerator from IWindow::EventType (see page 952), the library constructs an event with the appropriate message identifier.

presSpace

```
virtual IPresSpaceHandle presSpace() const;
```

Acquires and returns the presentation space handle (also known as graphics context, or GC) for the window.

Motif Information: 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.

rect

```
virtual IRectangle rect() const;
```

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.

In this release, your application must use OS/2 Presentation Manager coordinates.

refresh

```
virtual IWindow &refresh(Boolean refreshChildren = false);

virtual IWindow &refresh(
                    const IRectangle &invalidRectangle);
```

Depending on which version you use, this function invalidates and redraws either:

- A specified window rectangle.
- The entire window. If you specify true for *refreshChildren*, this function updates the children as well.

Motif Information: Motif ignores this function, 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.

releasePresSpace

```
virtual void releasePresSpace(
              const IPresSpaceHandle &aPresentationSpaceHandle) const;
```

Releases the window's presentation space handle.

Motif Information: 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.

resetActiveColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetActiveColor();
```

Resets the active color by undoing the previous set.

resetBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetBackgroundColor();
```

Resets the background color by undoing a previous set.

resetBorderColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetBorderColor();
```

Resets the border color by undoing a previous set.

resetDisabledBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetDisabledBackgroundColor();
```

Resets the disabled background color by undoing a previous set.

resetDisabledForegroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetDisabledForegroundColor();
```

Resets the disabled foreground color by undoing a previous set.

resetForegroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetForegroundColor();
```

Resets the foreground color by undoing a previous set.

resetHiliteBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetHiliteBackgroundColor();
```

Resets the highlight background color by undoing a previous set.

resetHiliteForegroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetHiliteForegroundColor();
```

Resets the highlight foreground color by undoing a previous set.

resetInactiveColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetInactiveColor();
```

Resets the inactive color by undoing a previous set.

resetMinimumSize

```
IWindow &resetMinimumSize();
```

Resets the minimum allowable size as if setMinimumSize had not been called.

resetShadowColor

Condition: (1) Ignored on AIX

```
virtual IWindow &resetShadowColor();
```

Resets the shadow color by undoing a previous set.

sendEvent

```
IEventResult sendEvent(
               unsigned long eventId,
               const IEventParameter1 &parm1 = 0,
               const IEventParameter2 &parm2 = 0) const;

IEventResult sendEvent(const IEvent &event) const;

IEventResult sendEvent(
               EventType eventType,
               const IEventParameter1 &parm1 = 0,
               const IEventParameter2 &parm2 = 0) const;
```

Sends an event constructed from the arguments to the window. If you specify an enumerator from IWindow::EventType (see page 952), the library constructs an event with the appropriate message identifier.

setActiveColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setActiveColor(const IColor &color);
```

Sets the active color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setAutoDeleteObject

```
IWindow &setAutoDeleteObject(Boolean autoDelete = true);
```

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.

setAutoDestroyWindow

```
IWindow &setAutoDestroyWindow(Boolean autoDestroy = false);
```

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.

setBackgroundColor

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

Sets the background color to the indicated color. The window area is identified by a system-defined presentation parameter value.

Motif Information: 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.

setBorderColor

```
virtual IWindow &setBorderColor(const IColor &color);
```

Sets the border color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setDefaultOrdering

```
static void setDefaultOrdering(SiblingOrder order);
```

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 (see page 953), the library creates new windows on top of their siblings.

setDisabledBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setDisabledBackgroundColor(
                    const IColor &color);
```

Sets the disabled background color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setDisabledForegroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setDisabledForegroundColor(
                    const IColor &color);
```

Sets the disabled foreground color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setExceptionFunction

```
static IWindow::ExceptionFn *setExceptionFunction(
                               IWindow::ExceptionFn *exceptionObject);
```

Stores an IWindow::ExceptionFn (see page 956) 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.

setFocus

```
virtual IWindow &setFocus();
```

Sets the input focus to the window.

setFont

```
virtual IWindow &setFont(const IFont &font);
```

Sets a new font to be used by the window.

setForegroundColor

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

Sets the foreground color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setHiliteBackgroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setHiliteBackgroundColor(
                    const IColor &color);
```

Sets the highlight background color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setHiliteForegroundColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setHiliteForegroundColor(
                    const IColor &color);
```

Sets the highlight foreground color to the indicated color. The window area is identified by a system-defined presentation parameter value.

setId

Condition: (1) Ignored on AIX

```
virtual IWindow &setId(unsigned long id);
```

Sets the window ID of the window.

setInactiveColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setInactiveColor(const IColor &color);
```

Sets the inactive color to the specified color. The window area is identified by a system-defined presentation parameter value.

setItemProvider

```
IWindow &setItemProvider(IDMItemProvider *dragProvider);
```

Sets a pointer to the drag item provider (IDMItemProvider) for this window.

Motif Information: AIX does not support this function.

setLayoutDistorted

Condition: (4) Typically for internal use

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

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

setMinimumSize

Condition: (4) Typically for internal use

```
IWindow &setMinimumSize(const ISize &size);
```

Sets the minimum allowable size of the window. In the User Interface Class Library, this applies only to windows on a canvas.

setOwner

```
virtual IWindow &setOwner(const IWindow *newOwner);
```

Changes the window's owner.

Motif Information: 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 which do not themselves support it.

setParent

Condition: (1) Ignored on AIX

```
virtual IWindow &setParent(const IWindow *newParent);
```

Changes the window's parent.

Motif Information: Motif does not support this function. The X Window System does not support any way to reparent widgets. Reparenting X windows separately from their widgets, although possible, is considered too dangerous for general use. If you are trying to reparent a window to effectively hide it, you must use IWindow::hide if you want system-independent, portable code.

setShadowColor

Condition: (1) Ignored on AIX

```
virtual IWindow &setShadowColor(const IColor &color);
```

Reset the shadow color by undoing a previous set.

shadowColor

Condition: (1) Ignored on AIX

```
virtual IColor shadowColor() const;
```

Returns the shadow color value of the window area, or the default if no color for the area has been set.

show

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

Makes the window visible.

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

showSourceEmphasis

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

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.

size

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

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.

In this release, your application must use OS/2 Presentation Manager coordinates.

sizeTo

```
virtual IWindow &sizeTo(const ISize &newSize);
```

Changes the size of the window, converting from the coordinates understood by this application to window system coordinates (if they are different).

Note: In this release, your application must use OS/2 Presentation Manager coordinates.

topHandle

Condition: (3) AIX only

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

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.

visibleRectangle

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

For controls like the drop-down combination box where the painted rectangle of the control is different than its actual rectangle, this function returns the painted rectangle. The default behavior of this function returns the actual rectangle.

windowWithHandle

```
static IWindow *windowWithHandle(
                  const IWindowHandle &windowHandle,
                  Boolean allThreads = true);
```

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.

windowWithOwner

```
static IWindow *windowWithOwner(
                  unsigned long id,
                  const IWindow *owner,
                  Boolean allThreads = true);
```

Returns the IWindow object for the specified ID 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.

windowWithParent

```
static IWindow *windowWithParent(
                  unsigned long id,
                  const IWindow *parent,
                  Boolean allThreads = true);
```

Returns the IWindow object for the specified ID 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::desktop for *parent*. This is a static function.

Figure 321. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
disableNotification	INotifier
enableNotification	INotifier
isEnabledForNotification	INotifier
notifyObservers	INotifier
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Protected Members

addHandler

Condition: (4) Typically for internal use

```
IWindow &addHandler(IHandler *newHandler);
```

Adds an IHandler object to the window.

addObserver

```
virtual IWindow &addObserver(
                    IObserver &anObserver,
                    const IEventData &userData = IEventData ( 0 ));
```

Adds an observer to the part's collection.

addRelatedHandleToWindowSet

Condition: (3) AIX only

Condition: (4) Typically for internal use

```
static void addRelatedHandleToWindowSet(
              IWindow *window,
              const IWindowHandle &windowHandle);
```

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.

addToWindowSet

Condition: (4) Typically for internal use

```
static void addToWindowSet(
              IWindow *window,
              const IWindowHandle &windowHandle);
```

Adds a window to the current thread's collection of windows. You must provide a valid window handle.

calcMinimumSize

Condition: (4) Typically for internal use

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

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.

color

Condition: (2) OS/2 only

```
IColor color(
            unsigned long colorArea,
            const IColor &defaultColor) const;
```

(2) OS/2 only

```
IColor color(unsigned long colorArea) const;
```

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 parameters are:

colorArea This is 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.

defaultColor
If you specify *defaultColor* and the window area has no color set, this color is returned. If you do not specify *defaultColor* and no color has been previously set, an IInvalidRequest exception is thrown.

Note: 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 setBackground.

create

Condition: (3) AIX only

```
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 ( ));
```

Condition: (2) OS/2 Only

```
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 ( ));
```

Calls the graphical presentation system to create an instance of a window. Generally, classes derived from IWindow which were implemented for Motif do not use this function. Instead, these classes use XtCreateWidget or an XmCreate&ellipse. helper routine.

defaultProcedure

```
IWindow &defaultProcedure(IEvent &event);
```

Provides default processing behavior for a window event. This function calls the window procedure that was returned from the window subclass during the call to IWindow::startHandlingEventsFor (see page 951).

Motif Information: 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

```
Boolean deleteIsInProcess() const;
```

If the IWindow object destructor has been called or the presentation window is being destroyed, true is returned.

dispatch

Condition: (1) Ignored on AIX

```
Boolean dispatch(IEvent &event);
```

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 (see page 946) 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 re-coordination is current.

Motif Information: 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.

isPrimaryWindow

```
Boolean isPrimaryWindow() const;
```

If this window has no owner and the parent is the desktop (root) window, true is returned.

isRelatedHandle

Condition: (3) AIX only

```
virtual Boolean isRelatedHandle(
                  const IWindowHandle &windowHandle) const;
```

Returns true if the specified handle is one of those which this IWindow derived class created in its constructor.

notificationHandler

```
IWindowNotifyHandler *notificationHandler() const;
```

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 request.

notifyObservers

```
virtual IWindow &notifyObservers(
                    const INotificationId &nId);
public:
virtual IWindow &notifyObservers(
                    const INotificationEvent &anEvent);
```

Notifies all observers in part's collection.

observerList

```
IObserverList &observerList() const;
```

Returns the collection of IObservers.

passEventToOwner

Condition: (3) AIX only

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

Determines if an event is passed on to the owner.

prepareForUse

Condition: (3) AIX only

```
virtual IWindow &prepareForUse(
                    const IWindowHandle &windowHandle);
```

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).

registerCallbacks

Condition: (3) AIX only

```
virtual void 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.

removeAllObservers

```
virtual IWindow &removeAllObservers();
```

Removes all observers from the part's collection.

removeFromWindowSet

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

Removes a window from the current thread's collection of windows.

removeHandler

```
IWindow &removeHandler(IHandler *oldHandler);
```

Removes a previously added handler.

removeObserver

```
virtual IWindow &removeObserver(IObserver &anObserver);
```

Removes an observer from the part's collection.

removeRelatedHandleFromWindowSet

Condition: (3) AIX only

Condition: (4) Typically for internal use

```
static void removeRelatedHandleFromWindowSet(
              IWindow *window,
              const IWindowHandle &windowHandle);
```

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.

resetColor

Condition: (2) OS/2 Only

```
IWindow &resetColor(unsigned long colorArea);
```

Resets the color of a window back to its default value by undoing any previous calls to setColor.

setColor

Condition: (2) OS/2 Only

```
IWindow &setColor(
            unsigned long colorArea,
            const IColor &color);
```

Sets the window area to the specified color. You can identify the window area using a system-defined presentation parameter value.

The parameters are:

colorArea This is 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 set to this color.

Note: 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 setBackground.

setNotificationHandler

```
IWindow &setNotificationHandler(
           IWindowNotifyHandler *notifyHandler);
```

Stores the window's notification handler. This handler captures window events and notifies observers of the window.

setStyle

Condition: (1) Ignored on AIX

```
virtual IWindow &setStyle(unsigned long style);
```

Sets the window style.

Motif Information: This function is ignored in AIX. The User Interface Class Library recommends using individual functions to set the various states of an IWindow::Style (see page 957) object, rather than the setStyle function. Use of this function will not result in portable code.

setWindowData

Condition: (2) OS/2 Only

```
IWindow &setWindowData(long index, unsigned short ushort);
```

Condition: (2) OS/2 Only

```
IWindow &setWindowData(long index, unsigned long ulong);
```

Sets the specified window word as an unsigned short or unsigned long. The result indicates whether it was successful.

Motif Information: 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.

startHandlingEventsFor

Condition: (4) Typically for internal use

```
IWindow &startHandlingEventsFor(
            unsigned long id,
            IWindow *parentDialog);
```

Condition: (4) Typically for internal use

```
IWindow &startHandlingEventsFor(
            const IWindowHandle &windowHandle);
```

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.

Motif Information: In X/Motif, the logic is somewhat different than for Presentation Manager and Windows. Motif does not use a single connection point like the Presentation Manager winproc. 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.

style

Condition: (1) Ignored on AIX

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

Returns an unsigned long representing the window's style.

Motif Information: 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 (see page 957) object, rather than the style function. Use of this function will not result in portable code.

unregisterCallbacks

Condition: (3) AIX only

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

Removes callbacks and X event handlers from this IWindow which were added by registerCallbacks. You *must* call Inherited::unregisterCallbacks so the handler objects can remove their processing.

windowULong

Condition: (2) OS/2 Only

```
unsigned long windowULong(long index) const;
```

Returns the specified window word as an unsigned long.

Motif Information: 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

Condition: (2) OS/2 Only

```
unsigned short windowUShort(long index) const;
```

Returns the specified window word as an unsigned short.

Motif Information: 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.

Figure 322. Inherited Protected Members

Member function	Class
addObserver	INotifier
notifyObservers	INotifier
observerList	INotifier
removeAllObservers	INotifier
removeObserver	INotifier

Note: * Application Support Class Library

Enumerations

EventType

```
public:
enum 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.

Layout

```
public:
enum 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.

SiblingOrder

```
public:
enum 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.

Chapter 217. IWindow::ChildCursor

Derivation

IBase
  IVBase
    IWindow::ChildCursor

Header File: iwindow.hpp

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.

Constructors

```
public:
ChildCursor(IWindow &parent);
```

You can construct objects of this class from a reference to the window over whose child windows you want to iterate.

Public Members

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

Marks the cursor as invalid.

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

If the cursor is in a valid area, true is returned.

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

Resets the cursor position to the first child window (in z-order).

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

Advances the cursor position to the next child window (in z-order).

Figure 323. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*

Figure 323. Inherited Public Members

Member function	Class
asString	IBase*
asString	IVBase*

© Copyright IBM Corp. 1994

Figure 323. Inherited Public Members

Member function	Class
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

Chapter 218. IWindow::ExceptionFn

Derivation

IBase
  IVBase
    IWindow::ExceptionFn

Header File: iwindow.hpp

Use the nested class IWindow::ExceptionFn to process exceptions occurring when events are dispatched by the IWindow (see page 924) dispatcher.

Public Members

handleException

```
virtual Boolean handleException(
                  IException &dispatcherException,
                  IEvent &exceptionEvent) = 0;
```

If an exception object is registered with IWindow (see page 924), this function is called when an uncaught exception occurs while dispatching events. If the exception situation was handled and it is safe to continue dispatching events, true is returned. If false is returned, the exception is thrown again.

Figure 324. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IVBase*
asString	IBase*
asString	IVBase*
operator <<	IBase*
operator <<	IVBase*

Note: * Application Support Class Library

© Copyright IBM Corp. 1994

Chapter 219. IWindow::Style

Derivation

IBase
  IBitFlag
    IWindow::Style

Header File: iwindow.hpp

The nested class IWindow::Style provides a set of valid window styles for the static member functions of the class IWindow (see page 924).

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 (see page 126).

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 the library should display on top of the other siblings, in z-order.

clipToParent
Allows a window to paint outside of its window boundary up to the window boundary of its parent.

disabled
Specifies that keyboard and mouse input are no longer dispatched to the window, preventing the window from being used.

noStyle
Sets all styles off.

saveBits
Optimizes the painting of a window by saving the screen image of the area under the window as a bit map, and then using the bit map to redraw the window when necessary.

© Copyright IBM Corp. 1994

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 (see page 950) to reset it after you create the window.

visible
Causes the window to be visible. In general, controls are constructed as visible and frame windows as invisible.

Motif Information: 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 (for example, the User Interface Class Library or its users) is not given any say in 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.

Public Members

Figure 325. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asString	IBase*
asUnsignedLong	IBitFlag
operator !=	IBitFlag
operator <<	IBase*
operator ==	IBitFlag

Note: * Application Support Class Library

Protected Members

Figure 326. Inherited Protected Members

Member function	Class
setValue	IBitFlag

Note: * Application Support Class Library

Chapter 220. IWindowHandle

Derivation

IBase
  IHandle
    IWindowHandle

Header File: ihandle.hpp

Objects of the IWindowHandle class access windows.

Motif Information: IWindowHandle is an alias for a widget data type.

Presentation Manager Information: IWindowHandle is an alias for the Presentation Manager Toolkit type HWND.

Constructors

```
public:
IWindowHandle(Value hwnd = 0);
```

Condition: (3) AIX only

```
public:
IWindowHandle(_WidgetRec *hwnd);
```

Condition: (3) AIX only

```
public:
IWindowHandle(int hwnd);
```

You can construct objects of this class from a window handle (a value of type IHandle::Value), which defaults to 0.

In Motif, construct objects of this class from an object of the X Toolkit type Widget.

Public Members

asUnsigned

```
unsigned asUnsigned() const;

unsigned long asUnsigned() const;
```

Returns the handle value as an unsigned value.

© Copyright IBM Corp. 1994

asUnsignedLongRef

```
unsigned long const &asUnsignedLongRef() const;
```

Returns a constant reference to an unsigned long value.

isValid

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

If the window handle is valid, true is returned.

operator Value

Condition: (2) OS/2 Only

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

Returns the IWindowHandle value.

operator _WidgetRec *

Condition: (3) AIX only

```
operator _WidgetRec *() const;
```

Returns the handle value as a native X-Toolkit window handle object.

postEvent

```
void postEvent(
       unsigned long eventId,
       const IEventParameter1 &parm1 = 0ul,
       const IEventParameter2 &parm2 = 0ul) const;
```

Posts an event constructed from the parameters to the window identified by this handle.

postEvents

```
void postEvents(
       unsigned long eventId,
       const IEventParameter1 &parm1 = 0ul,
       const IEventParameter2 &parm2 = 0ul,
       BroadcastTo value = descendants) const;
```

Posts one event to multiple windows.

sendEvent

```
IEventResult sendEvent(
               unsigned long eventId,
               const IEventParameter1 &parm1 = 0ul,
               const IEventParameter2 &parm2 = 0ul) const;
```

Sends an event constructed from the parameters to the window identified by this handle.

sendEvents

```
void sendEvents(
       unsigned long eventId,
       const IEventParameter1 &parm1 = 0ul,
       const IEventParameter2 &parm2 = 0ul,
       BroadcastTo value = descendants) const;
```

Broadcasts one event to multiple windows.

Figure 327. Inherited Public Members

Member function	Class
asDebugInfo	IBase*
asDebugInfo	IHandle
asString	IBase*
asString	IHandle
operator <<	IBase*

Note: * Application Support Class Library

Enumerations

BroadcastTo

```
public:
enum 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.

Appendix A. New Color Support

The User Interface Class Library has added new support for the setting, querying, and resetting of colors. In the previous releases of the User Interface Class Library, each class that supported the setting and querying of colors contained a ColorArea enumeration along with an implementation of the setColor and color functions. We no longer advise using this enumeration and the two corresponding functions and we have removed them from the interface. They still exist for backward compatibility; however, they might be removed in a future release.

The User Interface Class Library defines a new set of functions for easier color manipulation. For example, to set the background color of an IListBox control you now use the following:

```
listBox->setBackgroundColor(IColor::red);
```

To query the background color of the same listbox:

```
IColor color = listBox->backgroundColor();
```

To set the list box back to its default color:

```
listBox->resetBackgroundColor();
```

These new functions for handling colors mean you no longer have to use different ColorArea enumerators for each control class.

See the following tables for a complete cross reference for the new color functions:

Figure 328 on page 964
List of the old ColorArea enumerators and the new functions you use in their place.

Figure 329 on page 970
List of the new color functions supported in each class. We provide this table because the User Interface Class Library does not support all of the IWindow class's color functions in IWindow's derived classes.

© Copyright IBM Corp. 1994

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
IButton	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	disabledForeground	disabledForegroundColor setDisabledForegroundColor resetDisabledForegroundColor
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	border	borderColor setBorderColor resetBorderColor
ICanvas	background	backgroundColor setBackgroundColor resetBackgroundColor
IContainerControl	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	border	borderColor setBorderColor resetBorderColor
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	highlightBackground	hiliteBackgroundColor setHiliteBackgroundColor resetHiliteBackgroundColor
IEntryField	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	disabledForeground	disabledForegroundColor setDisabledForegroundColor resetDisabledForegroundColor

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	border	borderColor setBorderColor resetBorderColor
IFrameWindow	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	disableBackground	disabledBackgroundColor setDisabledBackgroundColor resetDisabledBackgroundColor
	frameBorder	borderColor setBorderColor resetBorderColor
	shadow	shadowColor setShadowColor resetShadowColor
	activeBorder	activeColor setActiveColor resetActiveColor
	inactiveBorder	inactiveColor setInactiveColor resetInctiveColor
IGroupBox	foreground	foregroundColor setForegroundColor resetForegroundColor
	border	borderColor setBorderColor resetBorderColor
IListBox	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	disabledForeground	disabledForegroundColor setDisabledForegroundColor resetDisabledForegroundColor

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	border	borderColor setBorderColor resetBorderColor
IMenu	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	disableForeground	disabledForegroundColor setDisabledForegroundColor resetDisabledForegroundColor
	disableBackground	disabledBackgroundColor setDisabledBackgroundColor resetDisabledBackgroundColor
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	highlightBackground	hiliteBackgroundColor setHiliteBackgroundColor resetHiliteBackgroundColor
	border	borderColor setBorderColor resetBorderColor
IMultiLineEdit	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	bordercolor	borderColor setBorderColor resetBorderColor
INotebook	pageBackground	backgroundPageColor setBackgroundPageColor resetBackgroundPageColor
	majorTabForeground	foregroundMajorColor setForegroundMajorColor resetForegroundMajorColor

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
	majorTabBackground	backgroundMajorColor setBackgroundMajorColor resetBackgroundMajorColor
	minorTabForeground	foregroundMinorColor setForegroundMinorColor resetForegroundMinorColor
	minorTabBackground	backgroundMinorColor setBackgroundMinorColor resetBackgroundMinorColor
	notebookWindowBackground	backgroundColor setBackgroundColor resetBackgroundColor
	notebookOutline	borderColor setBorderColor resetBorderColor
	statusLineText	foregroundColor setForegroundColor resetForegroundColor
	selectionCursor	hiliteBackgroundColor setHiliteBackgroundColor resetHiliteBackgroundColor
IOutlineBox	fillRegion	foregroundColor setForegroundColor resetForegroundColor
IProgressIndicator	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	border	borderColor setBorderColor resetBorderColor
IScrollBar	shaft	foregroundColor setForegroundColor resetForegroundColor
	scrollBox	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	border	borderColor setBorderColor resetBorderColor

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
ISetCanvas	background	backgroundColor setBackgroundColor resetBackgroundColor
ISpinButton	foreground	foregroundColor setForegroundColor resetForegroundColor
	disabledForeground	disabledForegroundColor setDisabledForegroundColor resetDisabledForegroundColor
	highlightForeground	hiliteForegroundColor setHiliteForegroundColor resetHiliteForegroundColor
	border	borderColor setBorderColor resetBorderColor
ISplitCanvas	splitBarEdge	splitBarEdgeColor setSplitBarEdgeColor resetSplitBarEdgeColor
	splitBarMiddle	splitBarMiddleColor setSplitBarMiddleColor resetSplitBarMiddleColor
IStaticText	foreground	foregroundColor setForegroundColor resetForegroundColor
	background	backgroundColor setBackgroundColor resetBackgroundColor
	fill	fillColor setFillColor resetFillColor
ITitle	activeFill	activeColor setActiveColor resetActiveColor
	inactiveFill	inactiveColor setInactiveColor resetInactiveColor
	activeTextForeground	activeTextForegroundColor setActiveTextForegroundColor resetActiveTextForegroundColor
	inactiveTextForeground	inactiveTextForegroundColor setInactiveTextForegroundColor resetInactiveTextForegroundColor

Figure 328. Enumerators to New Color Member Functions

Class	ColorArea enumerators	New Color Member Functions
	activeTextBackground	activeTextBackgroundColor setActiveTextBackgroundColor resetActiveTextBackgroundColor
	inactiveTextBackground	inactiveTextBackgroundColor setInactiveTextBackgroundColor resetInactiveTextBackgroundColor
	border	borderColor setBorderColor resetBorderColor
IViewPort	fillBackground	backgroundColor setBackgroundColor resetBackgroundColor

Figure 329. Classes to New Color Member Functions

Class	Query Color Member Function	Set Color Member Function	Reset Color Member Function
IButton	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
ICanvas	backgroundColor	setBackgroundColor	resetBackgroundColor
IContainerControl	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IEntryField	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IFrameWindow	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	shadowColor	setShadowColor	resetShadowColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	activeColor	setActiveColor	resetActiveColor
	inactiveColor	setInactiveColor	resetInactiveColor
	borderColor	setBorderColor	resetBorderColor
IGroupBox	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor

Figure 329. Classes to New Color Member Functions

Class	Query Color Member Function	Set Color Member Function	Reset Color Member Function
IListBox	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IMenu	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IMultiCellCanvas	backgroundColor	setBackgroundColor	resetBackgroundColor
IMultiLineEdit	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor
INotebook	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor
	foregroundColor	setForegroundColor	resetForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	backgroundPageColor	setBackgroundPageColor	resetBackgroundPageColor
	backgroundMajorColor	setBackgroundMajorColor	resetBackgroundMajorColor
	backgroundMinorColor	setBackgroundMinorColor	resetBackgroundMinorColor
	foregroundMajorColor	setForegroundMajorColor	resetForegroundMajorColor
	foregroundMinorColor	setForegroundMinorColor	resetForegroundMinorColor
IOutlineBox	foregroundColor	setForegroundColor	resetForegroundColor
IProgressIndicator	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IScrollBar	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	borderColor	setBorderColor	resetBorderColor

Figure 329. Classes to New Color Member Functions

Class	Query Color Member Function	Set Color Member Function	Reset Color Member Function
ISetCanvas	backgroundColor	setBackgroundColor	resetBackgroundColor
ISpinButton	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IStaticText	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor
ITitle	activeColor	setActiveColor	resetActiveColor
	inactiveColor	setInactiveColor	resetInactiveColor
	activeTextForegroundColor	setActiveTextForegroundColor	resetActiveTextForegroundColor
	activeTextBackgroundColor	setActiveTextBackgroundColor	resetActiveTextBackgroundColor
	inactiveTextForegroundColor	setInactiveTextForegroundColor	resetInactiveTextForegroundColor
	inactiveTextBackgroundColor	setInactiveTextBackgroundColor	resetInactiveTextBackgroundColor
	borderColor	setBorderColor	resetBorderColor
IViewPort	backgroundColor	setBackgroundColor	resetBackgroundColor
IWindow	foregroundColor	setForegroundColor	resetForegroundColor
	backgroundColor	setBackgroundColor	resetBackgroundColor
	borderColor	setBorderColor	resetBorderColor
	shadowColor	setShadowColor	resetShadowColor
	disabledForegroundColor	setDisabledForegroundColor	resetDisabledForegroundColor
	disabledBackgroundColor	setDisabledBackgroundColor	resetDisabledBackgroundColor
	hiliteForegroundColor	setHiliteForegroundColor	resetHiliteForegroundColor
	hiliteBackgroundColor	setHiliteBackgroundColor	resetHiliteBackgroundColor
	activeColor	setActiveColor	resetActiveColor
	inactiveColor	setInactiveColor	resetInactiveColor

Appendix B. Task and Samples Cross-Reference Table

The following table contains a list of User Interface Class Library tasks and cross-references them to the samples and examples in the *User Interface Class Library User's Guide* that show you how to complete the task.

Tasks	Sample or Example
Create a basic frame window with a line of text centered in the middle of the window	Hello World version 2
Dynamically define where text on a frame window is displayed	Hello World version 2
Display information at the bottom of a window	Hello World version 2
Title your application using default the default method	Hello World version 2
Title your application using a .RC file	Hello World version 2
Display application status after each event	Hello World version 3
Process menu bar items	Hello World version 3
Let the user perform functions using accelerator keys	Hello World version 4
Request text information from users using a dialog	Hello World versions 4, 5, and 6
Use a entry field on a frame window to update text on a screen	Hello World version 4
Use a push button on a frame window to accept or reject a change to a control	Hello World version 4
Display multiple controls in a frame window, client window, and canvas control	Hello World version 5
Display information to a user in a list	Hello World version 5
Let the user select text from a list of items to update text on a component using a list box	Hello World version 5
Perform an action when a user selects an item from a list	Hello World version 5
Add help to an application (requires use of IPFX/X on AIX)	Hello World version 5
Let the user select text from a list of items to update text on a component using a combo box	Hello World version 6
Show multiple components on a status line	Hello World version 6
Display a digital clock (long and short form)	Hello World version 6
Display the date (long and short form)	Hello World version 6
Let the user dynamically change the font of an application using a font dialog	Hello World version 6

© Copyright IBM Corp. 1994

Tasks	Sample or Example
Customize an application using a notebook control	Hello World version 6
Customize an application using a pop-up menu	Hello World version 6
Record a user's settings of an application	Hello World version 6
Customize a control	*User's Guide* (Chapter 5, "Setting Styles")
Save data from a user-edited control to a file	*User's Guide* (Chapter 6, "Viewing and Editing MLEs")
Display a list of choices to a user	*User's Guide* (Chapter 6, "Creating a Check Box")
Displaying a discrete set of choices to a user	*User's Guide* (Chapter 7, "Creating a Spin Button")
Arrange child controls in rows or columns	*User's Guide* (Chapter 8, "Creating a Set Canvas")
Let the user make a file selection using a file dialog control	*User's Guide* (Chapter 9, "Creating a File Dialog")
Display a set of related bit maps	*User's Guide* (Chapter 12, "Creating and Using Containers")
Manipulate a displayed set of related bit maps	*User's Guide* (Chapter 12, "Adding and Removing Objects")
Display detailed information related to an icon bit map	*User's Guide* (Chapter 12, "Defining the Details View Using Container Columns")
Process different events (time changes on a clock)	*User's Guide* (Chapter 14, "Extending Event Handling")

Appendix C. Handlers for Controls Cross-Reference Table

The following table lists the handlers that you can attach to User Interface Class Library control class objects.

Handler	Control
ICnrHandler	IContainerControl
ICnrMenuHandler	IContainerControl
ICommandHandler	IPushButton and owner IGraphicsPushButton IMenu and owner IFrameWindow
IEditHandler	IEntryField and owner IMultiLineEdit and owner IComboBox and owner ITextSpinButton and owner INumericSpinButton and owner IProgressIndicator and owner ISlider and owner
IEditVerifyHandler	IEntryField and owner IMultiLineEdit and owner IComboBox and owner ITextSpinButton and owner
IFileDialogHandler	IFileDialog
IFocusHandler	IContainerControl and owner IEntryField and owner IListBox and owner IMultiLineEdit and owner INumericSpinButton and owner ISlider and owner ITextSpinButton and owner
IFontDialogHandler	IFontDialog
IFrameHandler	IFrameWindow
IHelpHandler	IFrameWindow
IKeyboardHandler	IComboBox (AIX only) IEntryField IFrameWindow IMultiLineEdit INotebook
IMenuHandler	IFrameWindow IMenu Any control that owns a menu or that contains a pop-up menu
IMouseClickHandler	All visible controls
IPageHandler	INotebook and window owner
IPaintHandler	All visible controls

© Copyright IBM Corp. 1994

Handler	Control
IResizeHandler	All visible controls
IScrollHandler	IScrollBar (attached only to its owner)
ISelectHandler	ICheckBox and window owner IComboBox and window owner IContainerControl and window owner IListBox and window owner IRadioButton and window owner I3StateCheckBox and window owner
IShowListHandler	IComboBox and window owner
ISpinHandler	ITextSpinButton and window owner INumericSpinButton and window owner

Appendix D. Classes and Their Styles Cross-Reference Table

The following table shows the styles available for use by a specific class, and whether the User Interface Class Library for AIX release supports the style.

Class	Styles	AIX
IBaseSpinButton::Style	centerAlign	No
	classDefaultStyle	
	fastSpin	
	leftAlign	
	master	
	noBorder	
	readOnly	
	rightAlign	No
	servant	
IBitmapControl::Style	classDefaultStyle	
	sizeToBitmap	
IButton::Style	noPointerFocus	No
ICanvas::Style	classDefaultStyle	
ICheckBox::Style	autoSelect	No
	classDefaultStyle	
IComboBox::Style	anyData	No
	classDefaultStyle	
	dbcsData	No
	dropDownListType	
	dropDownType	
	horizontalScroll	
	mixedData	No
	sbcsData	No
	simpleType	
IContainerControl::Style	autoPosition	
	classDefaultStyle	
	extendedSelection	
	multipleSelection	
	readOnly	
	singleSelection	
	verifyPointers	
IControl::Style	group	No
	tabStop	No

© Copyright IBM Corp. 1994

Class	Styles	AIX
IEntryField::Style	anyData	
	autoScroll	
	autoTab	
	centerAlign	No
	classDefaultStyle	
	command	No
	dbcsData	No
	leftAlign	
	margin	
	mixedData	No
	readOnly	
	rightAlign	No
	sbcsData	
	unreadable	No
IFileDialog::Style	applyButton	
	classDefaultStyle	
	filter	No
	helpButton	
	includeEAS	No
	modeless	
	multiSelection	
	noStyle	
	preload	No
	selectableListbox	
IFontDialog::Style	applyButton	
	bitmapOnly	
	classDefaultStyle	
	fixedWidthOnly	
	helpButton	
	modeless	
	noStyle	
	noSynthesize	No
	proportionalOnly	
	resetButton	
	vectorOnly	

Class	Styles	AIX
IFrameWindow::Style	accelerator	
	alignNoAdjust	No
	animated	No
	appDBCSStatus	No
	border	
	classDefaultStyle	
	deferCreation	
	dialogBackground	No
	dialogBorder	
	hideButton	No
	horizontalScroll	
	maximizeButton	
	maximized	
	menuBar	
	minimizeButton	
	minimized	No
	minimizedIcon	No
	noMoveWithOwner	No
	shellPosition	No
	sizingBorder	
	systemMenu	
	systemModal	
	titleBar	
	verticalScroll	
	windowList	No
IGraphicPushButton::Style	classDefaultStyle	
	sizeToGraphic	
IGroupBox::Style	classDefaultStyle	
	radioGroupStyle	
IIconControl::Style	classDefaultStyle	
	sizeToIcon	
IListBox::Style	classDefaultStyle	
	drawItem	No
	extendedSelect	
	horizontalScroll	
	multipleSelect	
	noAdjustPosition	No
IMenu::Style	classDefaultStyle	
	verticalFlip	No
	noStyle	
IMenuBar::Style	classDefaultStyle	
	empty	
	wrapper	

Class	Styles	AIX
IMenuItem::Style	buttonSeparator	No
	classDefaultStyle	
	drawItem	No
	noStyle	
	postHelp	
	postSystemCommand	
	separator	
	split	
	splitWithSeparator	No
	unavailable	
IMessageBox::Style	abortRetryIgnoreButton	
	cancelButton	
	enterButton	
	enterCancelButton	
	okButton	
	okCancelButton	
	retryCancelButton	
	yesNoButton	
	yesNoCancelButton	
	defButton1	
	defButton2	
	defButton3	
	errorIcon	
	informationIcon	
	noIcon	
	queryIcon	
	warningIcon	
	applicationModal	
	systemModal	
	moveable	
IMultiCellCanvas::Style	classDefaultStyle	
	dragLines	No
	gridLines	No
IMultiLineEdit::Style	border	
	classDefaultStyle	
	horizontalScroll	
	ignoreTab	
	readOnly	
	verticalScroll	
	wordWrap	

Class	Styles	AIX
INotebook::Style	backPagesBottomLeft backPagesBottomRight backPagesTopLeft backPagesTopRight classDefaultStyle majorTabsBottom majorTabsLeft majorTabsRight majorTabsTop polygonTabs roundedTabs solidBinding spiralBinding squareTabs statusTextCenter statusTextLeft statusTextRight tabTextCenter tabTextLeft tabTextRight	
INumericSpinButton::Style	classDefaultStyle padWithZeros	
IOutlineBox::Style	backgroundFrame classDefaultStyle foregroundFrame halftoneFrame	 No
IProgressIndicator::Style	alignBottom alignCentered alignLeft alignRight alignTop classDefaultStyle handleDrawItem homeBottom homeLeft homeRight homeTop horizontal primaryScale1 primaryScale2 ribbonStrip snapToTickMark vertical	 No
IPushButton::Style	classDefaultStyle defaultButton help noBorder systemCommand	

Class	Styles	AIX
IRadioButton::Style	autoSelect	No
	classDefaultStyle	
	noCursorSelect	No
IScrollBar::Style	autoSize	
	classDefaultStyle	
	horizontal	
	vertical	
ISetCanvas::Style	bottomAlign	
	centerAlign	
	centerVerticalAlign	
	classDefaultStyle	
	horizontalDecks	
	leftAlign	
	packEven	
	packExpanded	
	packTight	
	rightAlign	
	topAlign	
	verticalDecks	
ISlider::Style	buttonsBottom	
	buttonsLeft	
	buttonsRight	
	buttonsTop	
	classDefaultStyle	
ISpinButton::Style	allCharacters	
	centerAlign	No
	classDefaultStyle	
	fastSpin	
	leftAlign	
	master	
	noBorder	
	numericOnly	
	padWithZeros	
	readOnly	
	rightAlign	No
	servant	
ISplitCanvas::Style	classDefaultStyle	
	horizontal	
	noSplitBars	
	vertical	

Class	Styles	AIX
IStaticText::Style	bottom	
	center	
	classDefaultStyle	
	fillBackground	
	halftone	No
	left	
	mnemonic	No
	right	
	strikeout	No
	top	
	underscore	No
	vertCenter	
	wordBreak	No
ITextSpinButton::Style	classDefaultStyle	
IViewPort::Style	alwaysHorizontalScrollBar	
	alwaysVerticalScrollBar	
	asNeededHorizontalScrollBar	
	asNeededVerticalScrollBar	
	classDefaultStyle	
	noHorizontalScrollBar	
	noVerticalScrollBar	
	noViewWindowFill	
IWindow::Style	clipChildren	No
	clipSiblings	No
	clipToParent	No
	disabled	
	noStyle	
	saveBits	
	synchPaint	
	visible	
	forcePaintOnResize	
	unrealized	
	managed	
	notMappedWhenManaged	

Appendix E. Classes and Header Files Cross-Reference Tables

This appendix contains the following cross-reference tables:

Figure 330 on page 984
List of classes, showing the header file where each class can be found.

Figure 331 on page 991
List of header files, showing the classes that each header file contains.

Figure 330. Classes to Header Files

Class Name	Header File
IAccelerator	iaccel.hpp
IAccelTblHandle	ihandle.hpp
IAnchorBlockHandle	ihandle.hpp
IApplication	iapp.hpp
IBaseSpinButton	ispinbas.hpp
IBaseSpinButton::Style	ispinbas.hpp
IBitFlag	ibitflag.hpp
IBitmapControl	ibmpctl.hpp
IBitmapControl::Style	ibmpctl.hpp
IBitmapHandle	ihandle.hpp
IButton	ibutton.hpp
IButton::Style	ibutton.hpp
ICanvas	icanvas.hpp
ICanvas::Style	icanvas.hpp
ICheckBox	icheckbx.hpp
ICheckBox::Style	icheckbx.hpp
ICnrControlList	icnrclst.hpp
ICnrEmphasisEvent	icnrevt.hpp
ICnrEnterEvent	icnrevt.hpp
ICnrEvent	icnrevt.hpp
ICnrHandler	icnrhdr.hpp
ICnrHelpEvent	icnrevt.hpp
ICnrMenuHandler	icnrmhdr.hpp
ICnrObjectSet	icnrolst.hpp
ICnrQueryDeltaEvent	icnrevt.hpp
ICnrScrollEvent	icnrevt.hpp
IColor	icolor.hpp

© Copyright IBM Corp. 1994

Figure 330. Classes to Header Files

Class Name	Header File
IComboBox	icombobx.hpp
IComboBox::Cursor	icombobx.hpp
IComboBox::Style	icombobx.hpp
ICommandEvent	icmdevt.hpp
ICommandHandler	icmdhdr.hpp
IContainerColumn	icnrcol.hpp
IContainerControl	icnrctl.hpp
IContainerControl::Attribute	icnrctl.hpp
IContainerControl::ColumnCursor	icnrctl.hpp
IContainerControl::CompareFn	icnrctl.hpp
IContainerControl::FilterFn	icnrctl.hpp
IContainerControl::Iterator	icnrctl.hpp
IContainerControl::ObjectCursor	icnrctl.hpp
IContainerControl::Style	icnrctl.hpp
IContainerControl::TextCursor	icnrctl.hpp
IContainerObject	icnrobj.hpp
IContextHandle	ihandle.hpp
IControl	icontrol.hpp
IControl::Style	icontrol.hpp
IControlEvent	ictlevt.hpp
ICritSec	icritsec.hpp
ICurrentApplication	iapp.hpp
ICurrentThread	ithread.hpp
IDeviceColor	icolor.hpp
IDisplayHandle	ihandle.hpp
IDynamicLinkLibrary	ireslib.hpp
IEditHandler	iedithdr.hpp
IEditVerifyEvent	iedvevt.hpp
IEditVerifyHandler	iedvhdr.hpp
IEntryField	ientryfd.hpp
IEntryField::Style	ientryfd.hpp
IEnumHandle	ihandle.hpp
IEvent	ievent.hpp
IEventData	ievtdata.hpp
IFileDialog	ifiledlg.hpp

Figure 330. Classes to Header Files

Class Name	Header File
IFileDialog::Settings	ifiledlg.hpp
IFileDialog::Style	ifiledlg.hpp
IFileDialogEvent	ifilehdr.hpp
IFileDialogHandler	ifilehdr.hpp
IFocusHandler	ifocshdr.hpp
IFont	ifont.hpp
IFontDialog	ifontdlg.hpp
IFontDialog::Settings	ifontdlg.hpp
IFontDialog::Style	ifontdlg.hpp
IFontDialogHandler	ifonthdr.hpp
IFrameEvent	iframevt.hpp
IFrameExtension	iframext.hpp
IFrameExtensions	iframext.hpp
IFrameFormatEvent	iframevt.hpp
IFrameHandler	iframhdr.hpp
IFrameWindow	iframe.hpp
IFrameWindow::Style	iframe.hpp
IGraphicPushButton	igraphbt.hpp
IGraphicPushButton::Style	igraphbt.hpp
IGroupBox	igroupbx.hpp
IGroupBox::Style	igroupbx.hpp
IGUIColor	icolor.hpp
IHandle	ihandle.hpp
IHandler	ihandler.hpp
IHelpErrorEvent	ihelpevt.hpp
IHelpHandler	ihelphdr.hpp
IHelpHypertextEvent	ihelpevt.hpp
IHelpMenuBarEvent	ihelpevt.hpp
IHelpNotifyEvent	ihelpevt.hpp
IHelpSubitemNotFoundEvent	ihelpevt.hpp
IHelpTutorialEvent	ihelpevt.hpp
IHelpWindow	ihelp.hpp
IHelpWindow::Settings	ihelp.hpp
IIconControl	iiconctl.hpp
IIconControl::Style	iiconctl.hpp

Figure 330. Classes to Header Files

Class Name	Header File
IInfoArea	iinfoa.hpp
IKeyboardEvent	ikeyevt.hpp
IKeyboardHandler	ikeyhdr.hpp
IListBox	ilistbox.hpp
IListBox::Cursor	ilistbox.hpp
IListBox::Style	ilistbox.hpp
IMenu	imenu.hpp
IMenu::Cursor	imenu.hpp
IMenu::Style	imenu.hpp
IMenuBar	imenubar.hpp
IMenuBar::Style	imenubar.hpp
IMenuEvent	imenuevt.hpp
IMenuHandler	imenuhdr.hpp
IMenuItem	imnitem.hpp
IMenuItem::Attribute	imnitem.hpp
IMenuItem::Style	imnitem.hpp
IMessageBox	imsgbox.hpp
IMessageBox::Style	imsgbox.hpp
IMessageQueueHandle	ihandle.hpp
IModuleHandle	ihandle.hpp
IMouseClickEvent	imcevt.hpp
IMouseClickHandler	imchdr.hpp
IMultiCellCanvas	imcelcv.hpp
IMultiCellCanvas::Style	imcelcv.hpp
IMultiLineEdit	imle.hpp
IMultiLineEdit::Style	imle.hpp
INotebook	inotebk.hpp
INotebook::Cursor	inotebk.hpp
INotebook::PageSettings	inotebk.hpp
INotebook::PageSettings::Attribute	inotebk.hpp
INotebook::Style	inotebk.hpp
INotifier	inotify.hpp
INumericSpinButton	ispinnum.hpp
INumericSpinButton::Style	ispinnum.hpp
IObjectWindow	iobjwin.hpp

Figure 330. Classes to Header Files

Class Name	Header File
IOutlineBox	ioutlbox.hpp
IOutlineBox::Style	ioutlbox.hpp
IPageEvent	ipageevt.hpp
IPageHandle	inotebk.hpp
IPageHandler	ipagehdr.hpp
IPageHelpEvent	ipageevt.hpp
IPageRemoveEvent	ipageevt.hpp
IPageSelectEvent	ipageevt.hpp
IPaintEvent	ipainevt.hpp
IPaintHandler	ipainhdr.hpp
IPair	ipoint.hpp
IPoint	ipoint.hpp
IPointerHandle	ihandle.hpp
IPopUpMenu	ipopmenu.hpp
IPresSpaceHandle	ihandle.hpp
IPrivateResource	ireslock.hpp
IProcedureAddress	iprocadr.hpp
IProcessId	ihandle.hpp
IProfile	iprofile.hpp
IProfile::Cursor	iprofile.hpp
IProfileHandle	ihandle.hpp
IProgressIndicator	islider.hpp
IProgressIndicator::Style	islider.hpp
IPushButton	ipushbut.hpp
IPushButton::Style	ipushbut.hpp
IRadioButton	iradiobt.hpp
IRadioButton::Style	iradiobt.hpp
IRange	ipoint.hpp
IRectangle	irect.hpp
IRefCounted	irefcnt.hpp
IReference	irefcnt.hpp
IRegionHandle	ihandle.hpp
IResizeEvent	isizeevt.hpp
IResizeHandler	isizehdr.hpp
IResource	ireslock.hpp

Figure 330. Classes to Header Files

Class Name	Header File
IResourceId	ireslib.hpp
IResourceLibrary	ireslib.hpp
IResourceLock	ireslock.hpp
IScrollBar	iscroll.hpp
IScrollBar::Style	iscroll.hpp
IScrollEvent	iscrlevt.hpp
IScrollHandler	iscrlhdr.hpp
ISelectHandler	iselhdr.hpp
ISemaphoreHandle	ihandle.hpp
ISetCanvas	isetcv.hpp
ISetCanvas::Style	isetcv.hpp
ISettingButton	isetbut.hpp
ISharedResource	ireslock.hpp
IShowListHandler	islhdr.hpp
ISize	ipoint.hpp
ISlider	islider.hpp
ISlider::Style	islider.hpp
ISpinButton	ispinbt.hpp
ISpinButton::Cursor	ispinbt.hpp
ISpinButton::Style	ispinbt.hpp
ISpinHandler	ispinhdr.hpp
ISplitCanvas	isplitcv.hpp
ISplitCanvas::Style	isplitcv.hpp
IStaticText	istattxt.hpp
IStaticText::Style	istattxt.hpp
IStringHandle	ihandle.hpp
ISubmenu	isubmenu.hpp
ISubmenu::Cursor	isubmenu.hpp
ISWP	iswp.hpp
ISWPArray	iswp.hpp
ISystemBitmapHandle	ihandle.hpp
ISystemMenu	isysmenu.hpp
ISystemPointerHandle	ihandle.hpp
ITextControl	itextctl.hpp
ITextSpinButton	ispintxt.hpp

Figure 330. Classes to Header Files

Class Name	Header File
ITextSpinButton::Cursor	ispintxt.hpp
ITextSpinButton::Style	ispintxt.hpp
IThread	ithread.hpp
IThread::Cursor	ithread.hpp
IThreadFn	ithread.hpp
IThreadId	ihandle.hpp
IThreadMemberFn	ithread.hpp
ITitle	ititle.hpp
IViewPort	ivport.hpp
IViewPort::Style	ivport.hpp
IWindow	iwindow.hpp
IWindow::ChildCursor	iwindow.hpp
IWindow::ExceptionFn	iwindow.hpp
IWindow::Style	iwindow.hpp
IWindowHandle	ihandle.hpp

Figure 331. Header Files to Classes

Header File	Classes
iaccel.hpp	IAccelerator
iapp.hpp	IApplication ICurrentApplication
ibitflag.hpp	IBitFlag
ibmpctl.hpp	IBitmapControl IBitmapControl::Style
ibutton.hpp	IButton IButton::Style
icanvas.hpp	ICanvas ICanvas::Style
icheckbx.hpp	ICheckBox ICheckBox::Style
icmdevt.hpp	ICommandEvent
icmdhdr.hpp	ICommandHandler
icnrclst.hpp	ICnrControlList
icnrcol.hpp	IContainerColumn
icnrctl.hpp	IContainerControl IContainerControl::Attribute IContainerControl::ColumnCursor IContainerControl::CompareFn IContainerControl::FilterFn IContainerControl::Iterator IContainerControl::ObjectCursor IContainerControl::Style IContainerControl::TextCursor
icnrevt.hpp	ICnrEmphasisEvent ICnrEnterEvent ICnrEvent ICnrHelpEvent ICnrQueryDeltaEvent ICnrScrollEvent
icnrhdr.hpp	ICnrHandler
icnrmhdr.hpp	ICnrMenuHandler
icnrobj.hpp	IContainerObject
icnrolst.hpp	ICnrObjectSet
icolor.hpp	IColor IDeviceColor IGUIColor
icombobx.hpp	IComboBox IComboBox::Cursor IComboBox::Style

Figure 331. Header Files to Classes

Header File	Classes
icontrol.hpp	IControl IControl::Style
icritsec.hpp	ICritSec
ictlevt.hpp	IControlEvent
iedithdr.hpp	IEditHandler
iedvevt.hpp	IEditVerifyEvent
iedvhdr.hpp	IEditVerifyHandler
ientryfd.hpp	IEntryField IEntryField::Style
ievent.hpp	IEvent
ievtdata.hpp	IEventData
ifiledlg.hpp	IFileDialog IFileDialog::Settings IFileDialog::Style
ifilehdr.hpp	IFileDialogEvent IFileDialogHandler
ifocshdr.hpp	IFocusHandler
ifont.hpp	IFont
ifontdlg.hpp	IFontDialog IFontDialog::Settings IFontDialog::Style
ifonthdr.hpp	IFontDialogHandler
iframe.hpp	IFrameWindow IFrameWindow::Style
iframevt.hpp	IFrameEvent IFrameFormatEvent
iframext.hpp	IFrameExtension IFrameExtensions
iframhdr.hpp	IFrameHandler
igraphbt.hpp	IGraphicPushButton IGraphicPushButton::Style
igroupbx.hpp	IGroupBox IGroupBox::Style

Figure 331. Header Files to Classes

Header File	Classes
ihandle.hpp	IAccelTblHandle IAnchorBlockHandle IBitmapHandle IContextHandle IDisplayHandle IEnumHandle IHandle IMessageQueueHandle IModuleHandle IPointerHandle IPresSpaceHandle IProcessId IProfileHandle IRegionHandle ISemaphoreHandle IStringHandle ISystemBitmapHandle ISystemPointerHandle IThreadId IWindowHandle
ihandler.hpp	IHandler
ihelp.hpp	IHelpWindow IHelpWindow::Settings
ihelpevt.hpp	IHelpErrorEvent IHelpHypertextEvent IHelpMenuBarEvent IHelpNotifyEvent IHelpSubitemNotFoundEvent IHelpTutorialEvent
ihelphdr.hpp	IHelpHandler
iiconctl.hpp	IIconControl IIconControl::Style
iinfoa.hpp	IInfoArea
ikeyevt.hpp	IKeyboardEvent
ikeyhdr.hpp	IKeyboardHandler
ilistbox.hpp	IListBox IListBox::Cursor IListBox::Style
imcelcv.hpp	IMultiCellCanvas IMultiCellCanvas::Style
imcevt.hpp	IMouseClickEvent
imchdr.hpp	IMouseClickHandler

Figure 331. Header Files to Classes

Header File	Classes
imenu.hpp	IMenu IMenu::Cursor IMenu::Style
imenubar.hpp	IMenuBar IMenuBar::Style
imenuevt.hpp	IMenuEvent
imenuhdr.hpp	IMenuHandler
imle.hpp	IMultiLineEdit IMultiLineEdit::Style
imnitem.hpp	IMenuItem IMenuItem::Attribute IMenuItem::Style
imsgbox.hpp	IMessageBox IMessageBox::Style
inotebk.hpp	INotebook INotebook::Cursor INotebook::PageSettings INotebook::PageSettings::Attribute INotebook::Style IPageHandle
inotify.hpp	INotifier
iobjwin.hpp	IObjectWindow
ioutlbox.hpp	IOutlineBox IOutlineBox::Style
ipageevt.hpp	IPageEvent IPageHelpEvent IPageRemoveEvent IPageSelectEvent
ipagehdr.hpp	IPageHandler
ipainevt.hpp	IPaintEvent
ipainhdr.hpp	IPaintHandler
ipoint.hpp	IPair IPoint IRange ISize
ipopmenu.hpp	IPopUpMenu
iprocadr.hpp	IProcedureAddress
iprofile.hpp	IProfile IProfile::Cursor
ipushbut.hpp	IPushButton IPushButton::Style

Figure 331. Header Files to Classes

Header File	Classes
iradiobt.hpp	IRadioButton IRadioButton::Style
irect.hpp	IRectangle
irefcnt.hpp	IRefCounted IReference
ireslib.hpp	IDynamicLinkLibrary IResourceId IResourceLibrary
ireslock.hpp	IPrivateResource IResource IResourceLock ISharedResource
iscrlevt.hpp	IScrollEvent
iscrlhdr.hpp	IScrollHandler
iscroll.hpp	IScrollBar IScrollBar::Style
iselhdr.hpp	ISelectHandler
isetbut.hpp	ISettingButton
isetcv.hpp	ISetCanvas ISetCanvas::Style
isizeevt.hpp	IResizeEvent
isizehdr.hpp	IResizeHandler
islhdr.hpp	IShowListHandler
islider.hpp	IProgressIndicator IProgressIndicator::Style ISlider ISlider::Style
ispinbas.hpp	IBaseSpinButton IBaseSpinButton::Style
ispinbt.hpp	ISpinButton ISpinButton::Cursor ISpinButton::Style
ispinhdr.hpp	ISpinHandler
ispinnum.hpp	INumericSpinButton INumericSpinButton::Style
ispintxt.hpp	ITextSpinButton ITextSpinButton::Cursor ITextSpinButton::Style
isplitcv.hpp	ISplitCanvas ISplitCanvas::Style

Figure 331. Header Files to Classes

Header File	Classes
istattxt.hpp	IStaticText IStaticText::Style
isubmenu.hpp	ISubmenu ISubmenu::Cursor
iswp.hpp	ISWP ISWPArray
isysmenu.hpp	ISystemMenu
itextctl.hpp	ITextControl
ithread.hpp	ICurrentThread IThread IThread::Cursor IThreadFn IThreadMemberFn
ititle.hpp	ITitle
ivport.hpp	IViewPort IViewPort::Style
iwindow.hpp	IWindow IWindow::ChildCursor IWindow::ExceptionFn IWindow::Style

Appendix F. Obsolete and Ignored Members Cross-Reference Tables

This appendix contains the following cross-reference tables:

Figure 332 on page 997
List of obsolete classes and members, and their replacements, if any.

Figure 333
List of classes and members that the User Interface Class Library for AIX does not support and subsequently ignores.

Figure 332. Obsolete Classes, Members, and their Replacements

Class	Member	Replacement
IContainerColumn	isHeadingReadOnly isReadOnly	isHeadingWriteable isWriteable
IContainerControl	detailObjectRectangle (both versions) isReadOnly	detailsObjectRectangle isWriteable
IContainerObject	iconOffset iconTextOffset isReadOnly	None None isWriteable
ICurrentThread	initializePM isPMInitialized terminatePM	initializeGUI isGUIInitialized terminateGUI
IFont	enum Direction bottomTop enum Direction leftRight enum Direction rightLeft enum Direction topBottom	enum Direction bottomToTop enum Direction leftToRight enum Direction rightToLeft enum Direction topToBottom
IHelpWindow	associateWindow helpForWindow	setAssociatedWindow None
IListBox	setHeight	setItemHeight
IMenu	addAt addNextAt isItemDisabled isVerticalFlip setVerticalFlip verticalFlip	add addAsNext isItemEnabled None None None
IMultiLineEdit	disableRefresh enableRefresh isReadOnly removeChangedFlag	disableUpdate enableUpdate isDataUpdateEnabled resetChangedFlag
IPushButton	disableBorder enableBorder	disableBorder enableBorder
ISpinButton		IBaseSpinButton INumericSpinButton ITextSpinButton

© Copyright IBM Corp. 1994

Figure 332. Obsolete Classes, Members, and their Replacements

Class	Member	Replacement
ISubmenu	addAt addNextAt	add addAsNext
IThread	autoInitPM defaultAutoInitPM setAutoInitPM setDefaultAutoInitPM	autoInitGUI defaultAutoInitGUI setAutoInitGUI setDefaultAutoInitGUI
IWindow	handleWithId isDisabled windowWithId	handleWithParent isEnabled windowWithOwner

A single asterisk (*) means this member is overloaded and you must look it up to determine which overload the AIX release ignores.

A double asterisk (**) means this class is not ported to the AIX platform.

Figure 333. Ignored members in MOTIF

Class	Members
I3StateCheckBox **	
I3StateCheckBox::Style **	
IAccelerator	handle isSet remove reset set *
IAccelTblHandle	
IAnchorBlockHandle	
IBaseSpinButton	alignment setAlignment
IButton	allowsMouseClickFocus disableMouseClickFocus enableMouseClickFocus highlight isHighlighted unhighlight
ICnrBeginEditEvent **	
ICnrDrawBackgroundEvent **	
ICnrDrawHandler **	
ICnrDrawItemEvent **	
ICnrEditEvent **	
ICnrEditHandler **	
ICnrEndEditEvent **	
ICnrEnterEvent	
ICnrHandler	cursoredChanged deltaReached inuseChanged windowScrolled
ICnrMenuHandler	addSourceEmphasis removeSourceEmphasis
ICnrQueryDeltaEvent **	atBottomDelta atFirstObject atLastObject atTopDelta ICnrQueryDeltaEvent ~ICnrQueryDeltaEvent

Figure 333. Ignored members in MOTIF

Class	Members
ICnrReallocStringEvent **	
ICnrScrollEvent	amount ICnrScrollEvent isHorizontal isLeftDetails isRightDetails isVertical ~ICnrScrollEvent
IContainerColumn	dataAttributes disableDataUpdate disableHeadingUpdate displayWidth enableDataUpdate enableHeadingUpdate headingIcon hideSeparators horizontalDataAlignment horizontalHeadingAlignment isHeadingIconHandle isHeadingReadOnly isHeadingString isHeadingWriteable isReadOnly isWriteable
IContainerColumn	justifyData justifyHeading setDataAttributes setDisplayWidth setHeadingIcon * setTitleAttributes showSeparators titleAttributes verticalDataAlignment verticalHeadingAlignment areDetailsViewTitlesVisible closeEdit collapse collapseTree columnUnderPoint convertToWorkspace currentEditColumn currentEditMLE currentEditObject cursoredObject

Figure 333. Ignored members in MOTIF

Class	Members
IContainerControl	detailObjectRectangle * detailsObjectRectangle * detailsTitleRectangle detailsViewPortOnWindow detailsViewPortOnWorkspace detailsViewSplit disableCaching disableDataUpdate disableDrawBackground disableDrawItem disableDrop disableTitleUpdate editColumnTitle editContainerTitle editObject enableCaching

Figure 333. Ignored members in MOTIF

Class	Members
IContainerControl	enableDataUpdate enableDrawBackground enableDrawItem enableDrop enableTitleUpdate expand expandTree hasMixedTargetEmphasis hasNormalTargetEmphasis hasOrderedTargetEmphasis hideSourceEmphasis * hideSplitBar hideTitle hideTitleSeparator iconRectangle iconSize isCachingEnabled isCollapsed isColumnRight isCursored isDrawBackgroundEnabled isDrawItemEnabled isDropOnAble isExpanded isInUse isReadOnly isRefreshOn isSource isTarget isTitleSeparatorVisible isTitleVisible isTitleWriteable isVisible * isWriteable lineSpacing

Figure 333. Ignored members in MOTIF

Class	Members
IContainerControl	moveIconTo objectAt * objectUnderPoint refresh * refreshAllContainers removeInUse scroll scrollDetailsHorizontally scrollHorizontally scrollToObject * scrollVertically setCursor setDetailsViewSplit setEditColumn setEditMLE setEditObject setIconSize setInUse setLineSpacing setMixedTargetEmphasis setNormalTargetEmphasis setOrderedTargetEmphasis setRefreshOff setRefreshOn setTitle * setTitleAlignment setTreeExpandIconSize showSourceEmphasis * showSplitBar showTitle showTitleSeparator splitBarOffset textRectangle title titleRectangle viewPortOnWindow viewPortOnWorkspace

Figure 333. Ignored members in MOTIF

Class	Members
IContainerObject	disableDataUpdate disableDrop enableDataUpdate enableDrop handleCursoredChange handleInuseChange iconOffset iconTextOffset isDropOnAble isInUse isReadOnly isRefreshOn isWriteable refresh removeInUse setInUse setRefreshOff setRefreshOn
IControl	disableGroup disableTabStop enableGroup enableTabStop isGroup isTabStop
ICurrentThread	remainingStack suspend waitFor waitForAllThreads waitForAnyThread
IDDEAcknowledgeEvent **	
IDDEAcknowledgeExecuteEvent **	
IDDEAcknowledgePokeEvent **	
IDDEActiveServer **	
IDDEActiveServerSet **	
IDDEBeginEvent **	
IDDEClientAcknowledgeEvent **	
IDDEClientConversation **	
IDDEClientEndEvent **	
IDDEClientHotLinkEvent **	
IDDEClientHotLinkSet **	
IDDEDataEvent **	
IDDEEndEvent **	
IDDEEvent **	

Figure 333. Ignored members in MOTIF

Class	Members
IDDEExecuteEvent **	
IDDEPokeEvent **	
IDDERequestDataEvent **	
IDDEServerAcknowledgeEvent **	
IDDEServerHotLinkEvent **	
IDDESetAcknowledgeInfoEvent **	
IDDETopicServer **	
IDeviceColor	
IDMCnrItem **	
IDMEFItem **	
IDMEvent **	
IDMHandler **	
IDMImage **	
IDMItem **	
IDMItemProvider **	
IDMItemProviderFor **	
IDMMLEItem **	
IDMOperation **	
IDMRenderer **	
IDMSourceBeginEvent **	
IDMSourceDiscardEvent **	
IDMSourceEndEvent **	
IDMSourceHandler **	
IDMSourceOperation **	
IDMSourcePrepareEvent **	
IDMSourcePrintEvent **	
IDMSourceRenderer **	
IDMSourceRenderEvent **	
IDMTargetDropEvent **	
IDMTargetEndEvent **	
IDMTargetEnterEvent **	
IDMTargetHandler **	
IDMTargetHelpEvent **	
IDMTargetLeaveEvent **	
IDMTargetOperation **	

Figure 333. Ignored members in MOTIF

Class	Members
IDMTargetRenderer **	
IDrawItemEvent **	
IDynamicLinkLibrary	
IEntryField	setAlignment setCharType
IEnumHandle	
IFileDialog::Settings	addDrive addFileType setInitialDrive setInitialFileType
IFont	isOutline isStrikeout isUnderscore setCharHeight setCharSize setCharWidth setDirection setFontAngle setFontShear setOutline setStrikeout setUnderscore
IFont::FaceNameCursor	
IFont::PointSizeCursor	
IFontDialog	fontWeight fontWidth
IFontDialog::Settings	setDisplayPS setPrinterPS
IFrameFormatEvent	clientRect IFrameFormatEvent setClientRect swpArray ~IFrameFormatEvent
IFrameHandler	deactivated draw format positionExtensions

Figure 333. Ignored members in MOTIF

Class	Members
IFrameWindow	addToWindowList beginFlashing borderHeight borderSize borderWidth endFlashing isFlashing isMaximized maximize maximizeRect minimizeRect nextShellRect removeFromWindowList restore restoreRect setBorderHeight setBorderSize * setBorderWidth setRestoreRect shareParentDBCSStatus usesDialogBackground
IGraphicPushButton	disableSizeToGraphic enableSizeToGraphic marginSize setMarginSize
IGroupBox	calcMinimumSize defaultStyle foregroundColor moveSizeTo position rect setDefaultStyle setText * size
IGroupBox::Style	
IGUIColor	
IHelpHandler	openLibrary showContents showCoverPage showHistory showIndex showPage showSearchList swapPage
IInfoArea	setResourceLibrary *

Figure 333. Ignored members in MOTIF

Class	Members
IListBox	disableNoAdjustPosition enableNoAdjustPosition isNoAdjustPosition
IListBoxDrawItemEvent **	
IListBoxDrawItemHandler **	
IListBoxDrawItemHandler::DrawFlag **	
IMenu	removeConditionalCascade setConditionalCascade
IMenuDrawItemEvent **	
IMenuDrawItemHandler **	
IMenuDrawItemHandler::DrawFlag **	
IMenuItem	isDrawItem isFramed isHighlighted isNoDismiss setDrawItem setFramed setHighlighted setNoDismiss
IMessageQueueHandle	
IModuleHandle	
IMultiCellCanvas	disableDragLines disableGridLines enableDragLines enableGridLines hasDragLines hasGridLines
IMultiLineEdit	isUndoable setTab undo
INotebook	hiliteBackgroundColor resetBackgroundMajorColor resetBackgroundMinorColor resetBackgroundPageColor resetForegroundMajorColor resetForegroundMinorColor setMajorTabSize setMinorTabSize setPageButtonSize setTabShape tabShape
INotebookDrawItemEvent **	
INotifier	

Figure 333. Ignored members in MOTIF

Class	Members
IPageHandler	help remove resize
IPageHelpEvent	helpWindow IPageHelpEvent * notebook pageHandle ~IPageHelpEvent
IPageRemoveEvent	IPageRemoveEvent * notebook pageWindow tabBitmap ~IPageRemoveEvent
IProcedureAddress	
IProfile	numberOfApplications
IProgressIndicator	disableDrawItem enableDrawItem isDrawItemEnabled setShaftPosition
IRadioButton	disableAutoSelect disableCursorSelect enableAutoSelect enableCursorSelect isAutoSelect isCursorSelect
IRegionHandle	
ISlider	addDetent detentPosition removeDetent setArmSize
ISliderDrawHandler **	
ISpinButton	alignment setAlignment
ISplitCanvas	resetSplitBarEdgeColor resetSplitBarMiddleColor setSplitBarEdgeColor splitBarEdgeColor
ISWP	
ISWPArray	
ISystemBitmapHandle **	
ISystemMenu	ISystemMenu *

Figure 333. Ignored members in MOTIF

Class	Members
IThread	adjustPriority defaultQueueSize defaultStackSize priorityClass priorityLevel queueSize resume setDefaultQueueSize setDefaultStackSize setPriority setQueueSize setStackSize stackSize suspend
ITitle	activeColor activeTextBackgroundColor activeTextForegroundColor borderColor inactiveColor inactiveTextBackgroundColor inactiveTextForegroundColor resetActiveTextBackgroundColor resetActiveTextForegroundColor resetInactiveTextBackgroundColor resetInactiveTextForegroundColor setActiveTextBackgroundColor setActiveTextForegroundColor setInactiveTextBackgroundColor setInactiveTextForegroundColor

Figure 333. Ignored members in MOTIF

Class	Members
IWindow	activeColor disabledBackgroundColor disabledForegroundColor dispatch hiliteBackgroundColor hiliteForegroundColor inactiveColor resetActiveColor resetBackgroundColor resetBorderColor resetDisabledBackgroundColor resetDisabledForegroundColor resetForegroundColor resetHiliteBackgroundColor resetHiliteForegroundColor resetInactiveColor resetShadowColor setActiveColor setDisabledBackgroundColor setDisabledForegroundColor setHiliteBackgroundColor setHiliteForegroundColor setId setInactiveColor setParent setShadowColor setStyle shadowColor style

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. 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 have a direct object of an abstract class. See also base class.

abstraction (data). See data abstraction.

access. An attribute that determines whether or not a class member is accessible in an expression or declaration.

access declaration. A declaration used to restore access to members of a base class.

access function. A function that returns information about the elements of a data 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.

ACU. Abstract code unit.

ambiguous derivation. A derivation where the class is derived from two or more base classes that have the same name.

anonymous union. A union that is declared within a structure or class and that does not have a name.

argument. In a function call, an expression that represents a value that the calling function passes to the function specified in the call. Synonymous with parameter.

B

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.

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.

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 User Interface Class Libraries example, a word boundary is a storage address evenly divisible by two.

built-in. A function which the compiler automatically puts inline instead of the function call unless the programmer specifies that it not do so.

C

catch block. A block associated with a try block that receives control when a C++ exception matching its argument is thrown.

cast. An expression that converts the type of the operand to a specified scalar data type (the operator).

child process. The new process created by a spawn or exec call.

class. An aggregate that can contain functions, types, and user-defined operators in addition to data. Classes can be defined hierarchically, allowing one class to be an expansion of another, and can restrict access to its members.

class hierarchy. A tree-like structure showing relationships among object classes. It places one

© Copyright IBM Corp. 1994

abstract class at the top (a base class) and one or more layers of less abstract classes (derived classes) below it.

class lattice. A structure that has an object class inheriting from multiple object classes. See also multiple inheritance.

class library. A collection of classes.

class member operator. An operator used to access class members through class objects or pointers to class objects. The class member operators are ., ->, .* and ->*.

client area object. 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.

Collection Classes. A set of classes that provide basic functions and can be used as base classes.

const. An attribute of a data object that declares the object cannot be changed.

constructor. A special class member function that has the same name as the class and is used to construct and possibly initialize class objects.

copy constructor. A constructor used to make a copy of a class object from another class object of the same class type.

critical section. Code that must be executed by one thread while all other threads in the process are suspended.

cursored emphasis. When the selection cursor is on a choice, that choice has cursored emphasis.

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.

data object. A storage area used to hold a value.

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. A description that makes an external object or function available to a function or a block.

declare. To identify the variable symbols to be used at preassembly time.

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 constructor. A constructor that takes no arguments, or a constructor for which all the arguments have default values.

delete. (1) A C++ keyword that identifies a free-storage deallocation operator. (2) A C++ operator used to destroy objects created by new.

derivation. The creation of a new or derived class from an existing or base class.

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.

doubleword. A sequence of bits or characters that comprises two computer words and can be addressed as a unit. For the IBM C/C++ Tools compiler, a doubleword is 32 bits (4 bytes).

E

elaborated type specifier. A specifier typically used in an incomplete class declaration to qualify types that are otherwise hidden.

encapsulation. The hiding of the internal representation of data objects and implementation details from the client program.

enumeration constant. An identifier that is defined in an enumeration and that has an associated 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.

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, have a variable number of parameters, and use _optlink linkage.

F

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 other class with the prefix friend.

function prolog. The code that appears at the beginning of a function and that links stack frames, saves registers, and allocates automatic storage.

G

global name space.. The first position in class names. IBM C Set ++ Class Libraries uses “I,” for “IBM.”

H

header file. A file that can contain system-defined control information or user data and generally consists of declarations.

I

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.

inheritance. (1) A mechanism by which an object class (derived class) can use the attributes, relationships, and methods 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. The assignment operator followed by an expression (or multiple expressions, for aggregate variables) used to set the initial value of a data object.

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 and the /Oi compiler option.

instance. Synonym for object, a particular instantiation of a data type.

instantiate. To create or generate a particular instance or object of a data type.

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.

intermediate code linker. A part of the IBM C/C++ Tools compiler that combines the information in all intermediate code files to improve optimization.

interrupt. A temporary suspension of a process caused by an external event, performed in such a way that the process can be resumed.

intrinsic function. A function supplied by a program as opposed to a function supplied by the compiler.

I/O Stream Library. A class library that provides the facilities to deal with many varieties of input and output.

item. A “proxy” for the object being manipulated.

K

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.

L

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.

linker. A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single executable program.

lvalue. An expression that represents a data object that can be both examined and altered.

M

member. (1) A data object in a structure or a union. (2) In C++, classes and structures can also contain functions and types as members.

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.

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 methods used by more than one base class.

See also inheritance and class lattice.

multitasking. A mode of operation that allows concurrent performance or interleaved execution of more than one task or program.

multithread. Pertaining to concurrent operation of more than one path of execution within a computer.

N

NaN. Not-a-Number.

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.

nonescaping key. (1) A key that allows a character to be typed without changing the imprint position. (2) A key that does not produce a character unless it is pressed in combination with another key.

Not-a-Number (NaN). A binary bit value for a floating-point type that is not equal to any other valid floating-point value, including itself. A NaN is typically the result of an operation that is not valid, such as division of zero by zero. A NaN can be either a signalling NaN (NaNS) that raises signals or exceptions, or a quiet NaN (NaNQ) that does not.

NULL. A pointer guaranteed not to point to a data object.

null character (\0). The ASCII or EBCDIC character with the hex value 00 (all bits turned off).

O

object. (1) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (2) A collection of data and methods (procedures) 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 methods. Each object in the class is said to be an instance of the class. (3) An instance of an object class consisting of (attributes) a data structure and operational methods. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and methods as other instances of the object class, though it has unique values assigned to its attributes.

object class. A template for defining the attributes and methods of an object. An object class can contain other object classes. An individual representation of an object class is called an object.

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.

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.

overloading. An object-oriented programming technique that allows you to redefine functions and most standard C++ operators when the functions and operators are used with class types.

P

pad. To fill unused positions in a field with data, usually 0's, 1's, or blanks.

parameter. See argument.

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 process. The program that originates the creation of other processes by means of spawn or exec function calls. See child process.

pointer. A variable that holds the address of a data object or function.

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)

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.

private. Pertaining to a class member that is accessible only to member functions and friends of that class.

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.

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 definition of = 0;.

R

resource file. A file that contains data used by an application, such as text strings and icons.

S

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.

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.

subclass. See derived class.

superclass. See base class and abstract class.

T

template. A family of classes or functions with variable types.

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.

thread. A unit of execution within a process.

throw expression. An argument to the C++ exception being thrown.

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.

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.

V

virtual function. A function of a class that is declared with the keyword virtual. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called. This is determined at run time.

volatile. An attribute of a data object that indicates the object is changeable. Any expression referring to a volatile object is evaluated immediately, for example, assignments.

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. For IBM C/C++ Tools, the character set is DBCS, and the value is 2 bytes.

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.

Index

A

accelerator
accessor functions
activated
  IFrameHandler 350
activeColor
  ITitle 909
  IWindow 927
activeTextBackgroundColor
  ITitle 909
activeTextForegroundColor
  ITitle 909
add
  IComboBox 112
  IListBox 462
  IMenu 482
  IMultiLineEdit 561
  ISpinButton 806
  ISubmenu 852
  ITextSpinButton 882
addAscending
  IComboBox 112
  IListBox 462
addAsFirst
  IComboBox 113
  IListBox 463
  ISpinButton 807
  ITextSpinButton 882
addAsLast
  IComboBox 113
  IListBox 463
  IMultiLineEdit 561
  ISpinButton 807
  ITextSpinButton 882
addAsNext
  IComboBox 113
  IListBox 464
  IMenu 482
  ISpinButton 808
  ISubmenu 853
  ITextSpinButton 882
addAt
  IMenu 482
  ISubmenu 853
addAtOffset
addAtOffset *(continued)*
  IMultiLineEdit 562
addBitmap
  IMenu 482
  ISubmenu 853
addBorder
  IBaseSpinButton 35
  IPushButton 700
addColumn
  IContainerControl 153
addDefaultHandler
  IFrameWindow 371
addDescending
  IComboBox 113
  IListBox 464
addDetent
  ISlider 797
addDrive
  IFileDialog::Settings 296
addExtension
  IFrameWindow 357
addFileType
  IFileDialog::Settings 297
addFirstPage
  INotebook 582
addHandler
  IWindow 946
addItem
  IMenu 483
  ISubmenu 854
addLastPage
  INotebook 583
addLibraries
  IHelpWindow 425
addLine
  IMultiLineEdit 562
addLineAsLast
  IMultiLineEdit 563
addNextAt
  IMenu 484
  ISubmenu 854
addObject
  IContainerControl 153
addObjectAfter
  IContainerControl 153
addObserver

© Copyright IBM Corp. 1994

addObserver *(continued)*
INotifier 613
IWindow 946
addOrReplaceElementWithKey
IProfile 674
addPageAfter
INotebook 583
addPageBefore
INotebook 583
addRef
IRefCounted 727
addRelatedHandleToWindowSet
IWindow 946
addSeparator
IMenu 484
ISubmenu 854
addSourceEmphasis
ICnrMenuHandler 96
IMenuHandler 511
addSubmenu
IMenu 484
addText
IMenu 484
ISubmenu 854
addToCell
IMultiCellCanvas 547
addToWindowList
IFrameWindow 358
addToWindowSet
IWindow 947
adjustPriority
IApplication 30
IThread 894
alignment
IBaseSpinButton 35, 41
IEntryField 259, 271
IProgressIndicator 684, 695
ISetCanvas 771, 778
ISpinButton 808, 818
IStaticText 839, 847
allObjectsDo
IContainerControl 153
allowChange
IEditVerifyEvent 252
allowsMouseClickFocus
IButton 57
amount
ICnrScrollEvent 102
anchorBlock
ICurrentThread 236
appContext
ICurrentThread 236
application classes
hierarchy 14
overview 2
applicationOrKeyAt
IProfile 674
applications
applyTo
IContainerControl::Iterator 202
appShell
ICurrentThread 236
area
IRectangle 720
areChildrenReversed
ICanvas 70
areDetailsViewTitlesVisible
IContainerControl 154
argc
ICurrentApplication 233
argv
ICurrentApplication 233
armPixelOffset
IProgressIndicator 684
armRange
IProgressIndicator 684
armSize
ISlider 797
armTickOffset
IProgressIndicator 685
arrangeIconView
IContainerControl 154
arrowDown
ISpinHandler 824
arrowUp
ISpinHandler 824
asDebugInfo
IApplication 31
ICurrentApplication 233
IDynamicLinkLibrary 246
IHandle 402
IHandler 405
IPair 651
IRectangle 720
IResourceId 738
IResourceLibrary 740
IThread 894
IWindow 927
asIndex
IComboBox::Cursor 125

asIndex *(continued)*
IListBox::Cursor 476
asLong
IEventData 285
asPixel
IColor 106
asPOINTL
IPoint 657
asRECTL
IRectangle 720
asRGBLong
IColor 106
asSIZEL
ISize 792
associateWindow
IHelpWindow 426
asString
IApplication 31
IDynamicLinkLibrary 247
IHandle 403
IHandler 405
IPair 652
IRectangle 720
IResourceId 739
IResourceLibrary 740
IThread 894
IWindow 927
asUnsigned
IWindowHandle 962
asUnsignedLong
IBitFlag 45
IEventData 285
asUnsignedLongRef
IWindowHandle 963
atBottomDelta
ICnrQueryDeltaEvent 99
atFirstObject
ICnrQueryDeltaEvent 99
atLastObject
ICnrQueryDeltaEvent 100
attachClient
IFrameWindow 371
attachedToId
IFrameExtension 341
attachTo
IFrameExtension 341
atTopDelta
ICnrQueryDeltaEvent 100
attribute
IMenuItem 515
attribute classes, hierarchy 15
attributes
IContainerControl 188
autoInitGUI
IThread 894
autoInitPM
IThread 894
avgCharWidth
IFont 314
avgLowercase
IFont 314
avgUppercase
IFont 314

B

backgroundColor
IButton 57
ICanvas 66
IEntryField 259
IFrameWindow 358
IListBox 464
IMenu 485
IMultiLineEdit 563
INotebook 584
IProgressIndicator 685
IStaticText 839
IWindow 927
backgroundMajorColor
INotebook 584
backgroundMinorColor
INotebook 584
backgroundPageColor
INotebook 584
baseRecord
IContainerObject 219
baseRecordSize
IContainerControl 188
baseRectFor
IFrameExtension 342
beginFlashing
IFrameWindow 358
beginUsingFont
IFont 314
binding
INotebook 585, 597
bitmap
IBitmapControl 48
IGraphicPushButton 384
IMenuItem 515

bitwise operators
blueMix
  IColor 106
books, related
  BookManager 10
  C and C++ 9
  C Set ++ for AIX 8
  C/C++ Tools for OS/2 8
  CUA 10
  OS/2 9
  OS/2 Technical Library 10
  OSF/Motif 10, 11
  style guides 10
  X Window System 11
borderColor
  ITitle 909
  IWindow 928
borderHeight
  IFrameWindow 358
borderSize
  IFrameWindow 359
borderWidth
  IFrameWindow 359
bottom
  IRectangle 720
bottomCenter
  IRectangle 721
bottomLeft
  IRectangle 721
bottomRight
  IRectangle 721
BroadcastTo
  IWindowHandle 964
buttonPressedId
  IFileDialog 290
  IFontDialog 326
buttonsPosition
  ISlider 797, 802

C

C Set ++ for AIX
  books 8
C/C++ Tools
  for OS/2 books 8
C++
  books 9
  standards 9
calcLimitSize
  IStaticText 846
calcMinimumSize
  IBaseSpinButton 40
  IBitmapControl 51
  ICanvas 70
  ICheckBox 77
  IComboBox 122
  IContainerControl 188
  IEntryField 269
  IGraphicPushButton 388
  IGroupBox 397
  IIconControl 439
  IInfoArea 449
  IListBox 473
  INotebook 596
  IProgressIndicator 693
  IPushButton 705
  IRadioButton 714
  IScrollBar 756
  ISpinButton 817
  IStaticText 846
  IViewPort 922
  IWindow 947
calcRect
  IFrameHandler 350
canvas classes
center
  IRectangle 721
centerAt
  IRectangle 721
centeredAt
  IRectangle 721
changed
  ICnrEmphasisEvent 82
char1
  IEventData 285
char2
  IEventData 285
char3
  IEventData 285
char4
  IEventData 286
character
  IKeyboardEvent 452
characterKeyPress
  IKeyboardHandler 459
characterSize
  IWindow 928
charType
  IEntryField 259, 271

charWidth
IFont 314
check box control
checkItem
IMenu 485
ISubmenu 855
childAt
IWindow 928
Classes
IAccelerator 24
IAccelTblHandle 28
IAnchorBlockHandle 29
IApplication 30
IBaseSpinButton 34
IBaseSpinButton::Style 42
IBitFlag 44
IBitmapControl 47
IBitmapControl::Style 53
IBitmapHandle 55
IButton 57
IButton::Style 63
ICanvas 65
ICanvas::Style 72
ICheckBox 73
ICheckBox::Style 79
ICnrControlList 81
ICnrEmphasisEvent 82
ICnrEnterEvent 85
ICnrEvent 87
ICnrHandler 89
ICnrHelpEvent 93
ICnrMenuHandler 95
ICnrObjectSet 98
ICnrQueryDeltaEvent 99
ICnrScrollEvent 102
IColor 105
IComboBox 110
IComboBox::Cursor 125
IComboBox::Style 128
ICommandEvent 130
ICommandHandler 133
IContainerColumn 137
IContainerControl 151
IContainerControl::Attribute 193
IContainerControl::ColumnCursor 196
IContainerControl::CompareFn 199
IContainerControl::FilterFn 201
IContainerControl::Iterator 202
IContainerControl::ObjectCursor 203
IContainerControl::Style 206
Classes *(continued)*
IContainerControl::TextCursor 208
IContainerObject 211
IContextHandle 221
IControl 222
IControl::Style 227
IControlEvent 229
ICritSec 231
ICurrentApplication 233
ICurrentThread 236
IDeviceColor 243
IDisplayHandle 245
IDynamicLinkLibrary 246
IEditHandler 249
IEditVerifyEvent 251
IEditVerifyHandler 255
IEntryField 258
IEntryField::Style 273
IEnumHandle 276
IEvent 277
IEventData 284
IFileDialog 288
IFileDialog::Settings 296
IFileDialog::Style 301
IFileDialogEvent 303
IFileDialogHandler 305
IFocusHandler 309
IFont 312
IFontDialog 324
IFontDialog::Settings 331
IFontDialog::Style 334
IFontDialogHandler 336
IFrameEvent 338
IFrameExtension 340
IFrameExtensions 346
IFrameFormatEvent 347
IFrameHandler 349
IFrameWindow 354
IFrameWindow::Style 378
IGraphicPushButton 382
IGraphicPushButton::Style 390
IGroupBox 392
IGroupBox::Style 398
IGUIColor 399
IHandle 402
IHandler 404
IHelpErrorEvent 407
IHelpHandler 410
IHelpHypertextEvent 415
IHelpMenuBarEvent 417

Classes *(continued)*
IHelpNotifyEvent 418
IHelpSubitemNotFoundEvent 420
IHelpTutorialEvent 422
IHelpWindow 424
IHelpWindow::Settings 433
IIconControl 435
IIconControl::Style 441
IInfoArea 443
IKeyboardEvent 451
IKeyboardHandler 457
IListBox 460
IListBox::Cursor 476
IListBox::Style 479
IMenu 481
IMenu::Cursor 497
IMenu::Style 499
IMenuBar 501
IMenuBar::Style 506
IMenuEvent 508
IMenuHandler 510
IMenuItem 514
IMenuItem::Attribute 523
IMenuItem::Style 525
IMessageBox 527
IMessageBox::Style 531
IMessageQueueHandle 534
IModuleHandle 536
IMouseClickEvent 537
IMouseClickHandler 541
IMultiCellCanvas 543
IMultiCellCanvas::Style 557
IMultiLineEdit 559
IMultiLineEdit::Style 579
INotebook 581
INotebook::Cursor 599
INotebook::PageSettings 602
INotebook::PageSettings::Attribute 607
INotebook::Style 609
INotifier 612
INumericSpinButton 614
INumericSpinButton::Style 621
IObjectWindow 623
IOutlineBox 626
IOutlineBox::Style 632
IPageEvent 634
IPageHandle 636
IPageHandler 637
IPageHelpEvent 640
IPageRemoveEvent 642

Classes *(continued)*
IPageSelectEvent 644
IPaintEvent 646
IPaintHandler 649
IPair 651
IPoint 657
IPointerHandle 659
IPopUpMenu 661
IPresSpaceHandle 666
IPrivateResource 668
IProcedureAddress 670
IProcessId 672
IProfile 673
IProfile::Cursor 677
IProfileHandle 679
IProgressIndicator 681
IProgressIndicator::Style 697
IPushButton 699
IPushButton::Style 707
IRadioButton 709
IRadioButton::Style 715
IRange 717
IRectangle 719
IRefCounted 727
IReference 729
IRegionHandle 731
IResizeEvent 732
IResizeHandler 734
IResource 736
IResourceId 738
IResourceLibrary 740
IResourceLock 746
IScrollBar 748
IScrollBar::Style 758
IScrollEvent 760
IScrollHandler 762
ISelectHandler 766
ISemaphoreHandle 769
ISetCanvas 770
ISetCanvas::Style 780
ISettingButton 783
ISharedResource 788
IShowListHandler 790
ISize 792
ISlider 794
ISlider::Style 803
ISpinButton 805
ISpinButton::Cursor 819
ISpinButton::Style 821
ISpinHandler 823

Classes *(continued)*
ISplitCanvas 826
ISplitCanvas::Style 836
IStaticText 838
IStaticText::Style 848
IStringHandle 851
ISubmenu 852
ISubmenu::Cursor 861
ISWP 863
ISWPArray 865
ISystemBitmapHandle 867
ISystemMenu 869
ISystemPointerHandle 874
ITextControl 876
ITextSpinButton 880
ITextSpinButton::Cursor 888
ITextSpinButton::Style 890
IThread 892
IThread::Cursor 902
IThreadFn 904
IThreadId 905
IThreadMemberFn 906
ITitle 908
IViewPort 916
IViewPort::Style 924
IWindow 926
IWindow::ChildCursor 956
IWindow::ExceptionFn 958
IWindow::Style 959
IWindowHandle 962
clear
IEntryField 260
IMultiLineEdit 563
clearBackground
IPaintEvent 647
clearLock
IResourceLock 747
click
IButton 58
client
IFrameWindow 359
client window
clientHandle
IFrameWindow 359
clientRect
IFrameFormatEvent 348
clientRectFor
IFrameWindow 359
clipboardHasTextFormat
ITextControl 876
close
IDynamicLinkLibrary 247
IFrameWindow 359
closed
IFrameHandler 351
closeEdit
IContainerControl 154
CMDSRC_MENU 514, 661
collapse
IContainerControl 154
collapseTree
IContainerControl 154
Color
color support 965
IColor 109
IWindow 947
color support, new 965
ColorArea
color support 965
column
ICnrHelpEvent 93
IContainerControl 189
columnAt
IContainerControl 154
columnCount
IContainerControl 154
columnInfo
IContainerColumn 147
columnUnderPoint
IContainerControl 155
columnWidth
IMultiCellCanvas 548
combination-box control
command
ICommandHandler 135
IFrameHandler 351
commandId
ICommandEvent 131
IHelpMenuBarEvent 417
commandType
IMenuItem 515, 521
communicationWindow
IHelpWindow 426
Conditions
Definitions 23
Notice 4
container
IContainerColumn 147
containerAttributes
IContainerControl 189

containerFromHandle
IContainerControl 155
containerId
ICnrEvent 88
containerInfo
IContainerControl 189
containerKey
IContainerControl 189
containerList
IContainerControl 189
containers
contains
IRectangle 721
containsApplication
IProfile 674
containsKeyName
IProfile 674
containsObject
IContainerControl 155
contentsWindow
IHelpWindow 426
control
IFrameExtension 342
controlHandle
IEvent 279
controlId
IControlEvent 229
IHelpNotifyEvent 418
controls
controlSelect
IHelpHandler 412
ControlType
IComboBox 124
controlWindow
IControlEvent 229
IEvent 279
conventions
data member names 5
file names 5
function arguments 6
function return types 6
global type names 5
maximum characters 5
member function names 5
numerations 5
type names 5
convertToWorkspace
IContainerControl 155
coord1
IPair 652
coord2
IPair 652
copy
IEntryField 260
IMultiLineEdit 564
copyObjectTo
IContainerControl 155
count
IComboBox 114
IListBox 464
coverPageWindow
IHelpWindow 426
cpp files, description 5
create
IFrameWindow 371
IWindow 948
current
IApplication 31
IContainerControl::ColumnCursor 196
IContainerControl::ObjectCursor 203
IContainerControl::TextCursor 208
INotebook::Cursor 599
IThread 894
currentEditColumn
IContainerControl 155
currentEditMLE
IContainerControl 156
currentEditObject
IContainerControl 156
currentGraphicType
IGraphicPushButton 384
currentId
IThread 895
currentPID
IApplication 31
cursor
IMenu 486
IMultiLineEdit 564
ISubmenu 855
cursoredChanged
ICnrHandler 90
cursoredObject
IContainerControl 156
cursors
cut
IEntryField 260
IMultiLineEdit 564

D

data member names, conventions 5
data types
  hierarchy 15
  overview 2
dataAsDate
  IContainerColumn 138
dataAsIcon
  IContainerColumn 138
dataAsNumber
  IContainerColumn 139
dataAsString
  IContainerColumn 139
dataAsTime
  IContainerColumn 139
dataAttributes
  IContainerColumn 147
DataSource
  IContainerColumn 148
DataStyle
  IContainerColumn 149
deactivated
  IFrameHandler 351
deckCount
  ISetCanvas 771
deckOrientation
  ISetCanvas 772, 778
decrementUseCount
  IContainerObject 219
defaultApplicationName
  IProfile 674
defaultAttribute
  IContainerControl 156
  IMenuItem 515
defaultAutoInitGUI
  IThread 895
defaultAutoInitPM
  IThread 895
defaultCell
  IMultiCellCanvas 548
defaultDataStyle
  IContainerColumn 139
defaultHeadingStyle
  IContainerColumn 139
defaultOrdering
  IFrameWindow 360
  IWindow 928
defaultProcedure
  IHandler 406
defaultProcedure *(continued)*
  IWindow 948
defaultPushButton
  ICanvas 67
  IWindow 928
defaultQueueSize
  IThread 895
defaultStackSize
  IThread 895
defaultStyle
  IBitmapControl 48
  ICanvas 67
  ICheckBox 74
  IComboBox 114
  IContainerControl 156
  IEntryField 260
  IFileDialog 290
  IFontDialog 326
  IFrameWindow 360
  IGraphicPushButton 384
  IGroupBox 393
  IIconControl 436
  IListBox 464
  IMenu 486
  IMenuBar 502
  IMenuItem 516
  IMultiCellCanvas 549
  IMultiLineEdit 565
  INotebook 585
  INumericSpinButton 616
  IOutlineBox 627
  IProgressIndicator 685
  IPushButton 701
  IRadioButton 710
  IScrollBar 749
  ISetCanvas 772
  ISlider 798
  ISpinButton 809
  ISplitCanvas 829
  IStaticText 839
  ITextSpinButton 883
  IViewPort 918
deleteAllObjects
  IContainerControl 156
deleteAt
  IMenu 486
  ISubmenu 855
deleteElementWithApplication
  IProfile 674

deleteElementWithKey
IProfile 674
deleteIsInProcess
IWindow 949
deleteItem
IMenu 486
ISubmenu 855
deleteSelectedObjects
IContainerControl 156
deltaReached
ICnrHandler 90
descendentsOf
IContainerControl 157
deselect
IComboBox 114
IListBox 465
ISettingButton 783
deselectAll
IComboBox 114
IListBox 465
desktopWindow
IWindow 928
detailObjectRectangle
IContainerControl 157
detailsObjectRectangle
IContainerControl 158
detailsTitleRectangle
IContainerControl 158
detailsViewPortOnWindow
IContainerControl 159
detailsViewPortOnWorkspace
IContainerControl 159
detailsViewSplit
IContainerControl 159
detentPosition
ISlider 798
deviceColor
IDeviceColor 243, 244
dialogs
Direction
IFont 322
disable
IEntryField 261
IHandler 405
IWindow 928
disableAutoScroll
IEntryField 261
disableAutoSelect
ICheckBox 74
IRadioButton 710
disableAutoSelect *(continued)*
ISettingButton 783
disableAutoTab
IEntryField 261
disableBorder
IPushButton 701
ISpinButton 809
disableCaching
IContainerControl 159
disableCommand
IEntryField 261
disableCursorSelect
IRadioButton 710
disableDataUpdate
IBaseSpinButton 35
IContainerColumn 139
IContainerControl 159
IContainerObject 212
IEntryField 261
IMultiLineEdit 565
disabledBackgroundColor
IFrameWindow 360
IMenu 486
IWindow 929
disableDefault
IPushButton 701
disabledForegroundColor
IButton 58
IMenu 487
IWindow 929
disableDragLines
IMultiCellCanvas 549
disableDrawBackground
IContainerControl 159
disableDrawItem
IContainerControl 159
IListBox 465
IProgressIndicator 685
disableDrop
IContainerControl 160
IContainerObject 213
disabledText
IInfoArea 444
disableExtendedSelect
IListBox 465
disableFastSpin
IBaseSpinButton 35
disableFillBackground
IStaticText 839

disableGridLines
IMultiCellCanvas 549
disableGroup
IControl 222
disableHalftone
IStaticText 840
disableHeadingUpdate
IContainerColumn 139
disableHelp
IPushButton 701
disableInsertMode
IEntryField 261
disableItem
IMenu 487
ISubmenu 855
disableMargin
IEntryField 261
disableMouseClickFocus
IButton 58
disableMultipleSelect
IListBox 465
disableNoAdjustPosition
IListBox 465
disableNotification
INotifier 612
IWindow 929
disableRefresh
IMultiLineEdit 565
disableRibbonStrip
IProgressIndicator 685
disableSizeToGraphic
IGraphicPushButton 384
disableSnapToTick
IProgressIndicator 685
disableStrikeout
IStaticText 840
disableSystemCommand
IPushButton 701
disableTabStop
IControl 223
disableTitleUpdate
IContainerControl 160
disableUnderscore
IStaticText 840
disableUpdate
IMultiLineEdit 565
IWindow 929
disableWordWrap
IMultiLineEdit 566
discard
IEntryField 261
IMultiLineEdit 566
dismiss
IFrameWindow 360
dispatch
IWindow 949
dispatchHandlerEvent
ICnrHandler 92
ICnrMenuHandler 97
ICommandHandler 135
IEditHandler 250
IEditVerifyHandler 257
IFileDialogHandler 306
IFocusHandler 310
IFontDialogHandler 337
IFrameHandler 353
IHandler 406
IHelpHandler 412
IInfoArea 449
IKeyboardHandler 459
IMenuHandler 512
IMouseClickHandler 542
IPageHandler 638
IPaintHandler 650
IResizeHandler 735
IScrollHandler 763
ISelectHandler 767
IShowListHandler 791
ISpinHandler 825
dispatchingWindow
IEvent 280
displaySize
ITextControl 876
displayWidth
IContainerColumn 140
distanceFrom
IPair 652
dotProduct
IPair 652
draw
IFrameHandler 351
drawSeparator
IFrameExtension 342
drawTab
IPageHandler 638
drawText
IPaintEvent 647

E

edit
IEditHandler 250
editColumnTitle
IContainerControl 160
editContainerTitle
IContainerControl 160
editObject
IContainerControl 160
editRegionHeight
IMultiLineEdit 566
editRegionWidth
IMultiLineEdit 566
editVerify
IEditVerifyHandler 257
elementAt
IComboBox 114
IListBox 465
IMenu 487
ISpinButton 809
ISubmenu 856
ITextSpinButton 883
elementWithKey
IProfile 675
emHeight
IFontDialog 326
Emphasis
IContainerObject 220
enable
IBaseSpinButton 35
IEntryField 262
IHandler 405
ISpinButton 809
IWindow 929
enableAutoScroll
IEntryField 262
enableAutoSelect
ICheckBox 74
IRadioButton 710
ISettingButton 783
enableAutoTab
IEntryField 262
enableBorder
IPushButton 701
ISpinButton 809
enableCaching
IContainerControl 160
enableCommand
IEntryField 262
enableCursorSelect
IRadioButton 710
enableDataUpdate
IBaseSpinButton 35
IContainerColumn 140
IContainerControl 161
IContainerObject 213
IEntryField 262
IMultiLineEdit 566
enableDefault
IPushButton 701
enableDragLines
IMultiCellCanvas 549
enableDrawBackground
IContainerControl 161
enableDrawItem
IContainerControl 161
IListBox 465
IProgressIndicator 685
enableDrop
IContainerControl 161
IContainerObject 213
enableExtendedSelect
IListBox 466
enableFastSpin
IBaseSpinButton 35
enableFillBackground
IStaticText 840
enableGridLines
IMultiCellCanvas 549
enableGroup
IControl 223
enableHalftone
IStaticText 840
enableHeadingUpdate
IContainerColumn 140
enableHelp
IPushButton 701
enableInsertMode
IEntryField 262
enableItem
IMenu 487
ISubmenu 856
enableMargin
IEntryField 263
enableMouseClickFocus
IButton 58
enableMultipleSelect
IListBox 466

enableNoAdjustPosition
IListBox 466
enableNotification
IComboBox 114
IContainerControl 161
IEntryField 263
IFrameWindow 360
IListBox 466
IMenu 487
IMultiLineEdit 566
INotebook 585
INotifier 612
IScrollBar 749
ISettingButton 784
ITitle 909
IWindow 929
enableRefresh
IMultiLineEdit 566
enableRibbonStrip
IProgressIndicator 686
enableSizeToGraphic
IGraphicPushButton 384
enableSnapToTick
IProgressIndicator 686
enableStrikeout
IStaticText 841
enableSystemCommand
IPushButton 702
enableTabStop
IControl 223
enableTitleUpdate
IContainerControl 162
enableUnderscore
IStaticText 841
enableUpdate
IMultiLineEdit 567
IWindow 930
enableWordWrap
IMultiLineEdit 567
endFlashing
IFrameWindow 360
endPosition
IEditVerifyEvent 252
endUsingFont
IFont 315
enter
ICnrHandler 90
ISelectHandler 767
enterPressed
ICnrEnterEvent 85
entry field control
EnumerationOrder
IContainerControl 191
EOLFormat
IMultiLineEdit 578
error
IHelpErrorEvent 407
ErrorType
IHelpErrorEvent 408
event
object 3
overview 3
event classes, hierarchy 16
event handler
eventId
IEvent 280
eventType
IEvent 280, 283
IWindow 954
exceptionFunction
IWindow 930
exit
ICurrentApplication 233
ICurrentThread 237
expand
IContainerControl 162
expandBy
IRectangle 721
expandedBy
IRectangle 721
expandForText
ISetCanvas 777
expandTree
IContainerControl 162
exportSelectedTextToFile
IMultiLineEdit 567
exportToFile
IMultiLineEdit 567
extensions
IFrameWindow 372
externalLeading
IFont 315
IFontDialog 326

F

faceNameAt
IFont 315
fattrs
IFont 315

file dialog
fileLength
IFileDialogEvent 303
fileName
IDynamicLinkLibrary 247
IFileDialog 290
IFileDialog::Settings 297
IFileDialogEvent 303
IResourceLibrary 741
files
class library conventions 5
hpp files 5
inl files 5
lib files 5
maximum characters 5
fillColor
IStaticText 841
Filter
IComboBox::Cursor 127
IContainerControl 162
IFileDialogHandler 306
IListBox::Cursor 478
filterName
IFileDialogHandler 307
findExtension
IFrameWindow 372
findExtensions
IFrameWindow 372
first
IContainerControl::ColumnCursor 196
IContainerControl::ObjectCursor 203
IContainerControl::TextCursor 209
INotebook::Cursor 599
firstPage
INotebook 585
fixedSize
IFrameExtension 342
fixupChildren
ICanvas 70
flags
ISWP 863
flagsFrom
IFrameWindow 373
font
IWindow 930
font dialog
fontFamily
IFontDialog 326
fontmetrics
IFont 315
fonts
fontWeight
IFontDialog 326
fontWidth
IFontDialog 326
foregroundColor
IBaseSpinButton 36
IButton 58
IEntryField 263
IGroupBox 393
IMenu 488
IMultiLineEdit 568
IOutlineBox 627
IProgressIndicator 686
IScrollBar 750
ISpinButton 809
IStaticText 841
IWindow 930
foregroundMajorColor
INotebook 585
foregroundMinorColor
INotebook 585
format
IFrameHandler 351
frame
IFrameEvent 338
frame extensions
frame window
frameRectFor
IFrameWindow 360
function arguments, conventions 6
function return types, conventions 6

G

global names, conventions 5
gotFocus
IFocusHandler 310
GraphicType
IGraphicPushButton 389
graphicWindow
IGraphicPushButton 384
greenMix
IColor 106
groupPad
ISetCanvas 772

H

h files, description 5
handle
  IAccelerator 25
  IDynamicLinkLibrary 247
  IEvent 281
  IHelpWindow 426
  IPrivateResource 669
  IProfile 675
  IResource 737
  IResourceLibrary 741
  ISharedResource 789
  ISubmenu 856
  IWindow 930
handleCursoredChange
  IContainerObject 213
handleError
  IHelpHandler 412
handleEventsFor
  ICnrHandler 90
  ICnrMenuHandler 95
  IFileDialogHandler 306
  IHandler 405
  IHelpHandler 411
  IInfoArea 444
  IMenuHandler 511
handleException
  IWindow 930
  IWindow::ExceptionFn 958
handleFor
  IFrameWindow 361
handleInuseChange
  IContainerObject 213
handleOnTop
  IMouseClickEvent 538
handleOpen
  IContainerObject 214
handler
  overview 3
handler classes, hierarchy 17
handlers for controls 977
handleSelectedChange
  IContainerObject 214
handleTreeCollapse
  IContainerObject 214
handleTreeExpand
  IContainerObject 214
handleWithParent
  IWindow 930
hasBorder
  IBaseSpinButton 36
hasChanged
  IEntryField 263
hasDragLines
  IMultiCellCanvas 549
hasFillBackground
  IStaticText 841
hasFocus
  IWindow 931
hasGridLines
  IMultiCellCanvas 550
hasHorizontalSeparator
  IContainerColumn 140
hasMixedTargetEmphasis
  IContainerControl 162
hasNormalTargetEmphasis
  IContainerControl 162
hasOrderedTargetEmphasis
  IContainerControl 162
hasSelectedText
  IEntryField 263
  IMultiLineEdit 568
hasVerticalSeparator
  IContainerColumn 141
headingIcon
  IContainerColumn 141
HeadingStyle
  IContainerColumn 150
headingText
  IContainerColumn 141
height
  IRectangle 722
  ISize 793
Hello World samples
Hello World version 1
Hello World version 2
Hello World version 3
Hello World version 4
Hello World version 5
Hello World version 6
help
  ICnrHandler 90
  IPageHandler 638
helpForWindow
  IHelpWindow 426
helpId
  IContainerColumn 141
  IContainerObject 214

HelpType
IHelpWindow 432
helpUndefined
IHelpHandler 412
helpWindow
IHelpWindow 426
IPageHelpEvent 641
hide
IContainerColumn 141
IContainerObject 214
IHelpWindow 427
IWindow 931
hideDetailsViewTitles
IContainerControl 163
hideList
IComboBox 114
hideObject
IContainerControl 163
hidePanelIds
IHelpWindow 427
hideSeparators
IContainerColumn 141
hideSourceEmphasis
IContainerControl 163
IWindow 931
hideSplitBar
IContainerControl 163
hideTitle
IContainerControl 163
hideTitleSeparator
IContainerControl 163
hideTreeLine
IContainerControl 164
hierarchy
application classes 14
data types and attributes 15
event classes 16
handler classes 17
setting classes 18
style classes 18
support classes 19
window classes 20
highHighByte
IEventData 286
highlight
IButton 58
highLowByte
IEventData 286
highNumber
IEventData 286
hiliteBackgroundColor
IButton 59
IMenu 488
INotebook 585
IWindow 931
hiliteForegroundColor
IButton 59
IMenu 488
IScrollBar 750
IWindow 931
homePosition
IProgressIndicator 686, 695
HorizontalAlignment
IContainerColumn 148
horizontalDataAlignment
IContainerColumn 141
horizontalHeadingAlignment
IContainerColumn 142
horizontalScrollBar
IViewPort 918
hpp files, description 5
hwndAllocation
IContainerControl 189
hypertextSelect
IHelpHandler 412

I

IAccelerator 24
IAccelTblHandle 28
IAnchorBlockHandle 29
IApplication 30
IBaseSpinButton 34
IBaseSpinButton::Style 42
IBitFlag 44
IBitmapControl 47
IBitmapControl::Style 53
IBitmapHandle 55
ibmcl.cat 5
IButton 57
IButton::Style 63
ICanvas 65
ICanvas::Style 72
ICheckBox 73
ICheckBox::Style 79
ICnrControlList 81
ICnrEmphasisEvent 82
ICnrEnterEvent 85
ICnrEvent 87

ICnrHandler 89
ICnrHelpEvent 93
ICnrMenuHandler 95
ICnrObjectSet 98
ICnrQueryDeltaEvent 99
ICnrScrollEvent 102
IColor 105
IComboBox 110
IComboBox::Cursor 125
IComboBox::Style 128
ICommandEvent 130
ICommandHandler 133
icon
  IContainerObject 215
  IFrameWindow 361
  IGraphicPushButton 385
  IIconControl 436
iconOffset
  IContainerObject 215
iconRectangle
  IContainerControl 164
iconSize
  IContainerControl 164
IContainerColumn 137
IContainerControl 151
IContainerControl::Attribute 193
IContainerControl::ColumnCursor 196
IContainerControl::CompareFn 199
IContainerControl::FilterFn 201
IContainerControl::Iterator 202
IContainerControl::ObjectCursor 203
IContainerControl::Style 206
IContainerControl::TextCursor 208
IContainerObject 211
iconText
  IContainerObject 215
IContextHandle 221
iconTextOffset
  IContainerObject 215
IControl 222
IControl::Style 227
IControlEvent 229
ICritSec 231
ICurrentApplication 233
ICurrentThread 236
id
  IApplication 31
  ICurrentThread 237
  IHelpHypertextEvent 415
  IMenuItem 516
id *(continued)*
  IResourceId 739
  IThread 896
  IWindow 931
Identifier
  ISystemBitmapHandle 868
  ISystemPointerHandle 875
IDeviceColor 243
IDisplayHandle 245
IDynamicLinkLibrary 246
IEditHandler 249
IEditVerifyEvent 251
IEditVerifyHandler 255
IEntryField 258
IEntryField::Style 273
IEnumHandle 276
IEvent 277
  object 3
  overview 3
IEventData 284
IFileDialog 288
IFileDialog::Settings 296
IFileDialog::Style 301
IFileDialogEvent 303
IFileDialogHandler 305
IFocusHandler 309
IFont 312
IFontDialog 324
IFontDialog::Settings 331
IFontDialog::Style 334
IFontDialogHandler 336
IFrameEvent 338
IFrameExtension 340
IFrameExtensions 346
IFrameFormatEvent 347
IFrameHandler 349
IFrameWindow 354
IFrameWindow::Style 378
IGraphicPushButton 382
IGraphicPushButton::Style 390
IGroupBox 392
IGroupBox::Style 398
IGUIColor 399
IHandle 402
IHandler 404
IHelpErrorEvent 407
IHelpHandler 410
IHelpHypertextEvent 415
IHelpMenuBarEvent 417

IHelpNotifyEvent 418
IHelpSubitemNotFoundEvent 420
IHelpTutorialEvent 422
IHelpWindow 424
IHelpWindow::Settings 433
IIconControl 435
IIconControl::Style 441
IInfoArea 443
IKeyboardEvent 451
IKeyboardHandler 457
IListBox 460
IListBox::Cursor 476
IListBox::Style 479
IMenu 481
IMenu::Cursor 497
IMenu::Style 499
IMenuBar 501
IMenuBar::Style 506
IMenuEvent 508
IMenuHandler 510
IMenuItem 514
IMenuItem::Attribute 523
IMenuItem::Style 525
IMessageBox 527
IMessageBox::Style 531
IMessageQueueHandle 534
immediateDescendentsOf
IContainerControl 164
IModuleHandle 536
IMouseClickEvent 537
IMouseClickHandler 541
importFromFile
IMultiLineEdit 568
IMultiCellCanvas 543
IMultiCellCanvas::Style 557
IMultiLineEdit 559
IMultiLineEdit::Style 579
inactiveColor
ITitle 910
IWindow 932
inactiveText
IInfoArea 444
inactiveTextBackgroundColor
ITitle 910
inactiveTextForegroundColor
ITitle 910
includes
IRange 717
incrementUseCount
IContainerObject 219
index
IColor 106
IMenuItem 516
indexOf
ISWPArray 865
indexWindow
IHelpWindow 427
information area
informationFor
IInfoArea 449
initialDrive
IFileDialog::Settings 297
initialFileType
IFileDialog::Settings 297
initialize
IBaseSpinButton 40
IContainerControl 164
IContainerObject 219
IEntryField 269
IFrameWindow 373
IProgressIndicator 693
initializeGUI
ICurrentThread 237
initializePM
ICurrentThread 237
inl files, description 5
INotebook 581
INotebook::Cursor 599
INotebook::PageSettings 602
INotebook::PageSettings::Attribute 607
INotebook::Style 609
INotifier 612
inputTypeAllowed
ISpinButton 809
integerWithKey
IProfile 675
internalLeading
IFont 315
intersects
IRectangle 722
INumericSpinButton 614
INumericSpinButton::Style 621
inuseChanged
ICnrHandler 91
invalidate
IComboBox::Cursor 125
IContainerColumn 147
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209

invalidate *(continued)*
IListBox::Cursor 476
IMenu::Cursor 497
INotebook::Cursor 600
IProfile::Cursor 677
ISpinButton::Cursor 819
ISubmenu::Cursor 861
ITextSpinButton::Cursor 888
IThread::Cursor 902
IWindow::ChildCursor 956
IObjectWindow 623
IOutlineBox 626
IOutlineBox::Style 632
IPageEvent 634
IPageHandle 636
IPageHandler 637
IPageHelpEvent 640
IPageRemoveEvent 642
IPageSelectEvent 644
IPaintEvent 646
IPaintHandler 649
IPair 651
IPoint 657
IPointerHandle 659
IPopUpMenu 661
IPresSpaceHandle 666
IPrivateResource 668
IProcedureAddress 670
IProcessId 672
IProfile 673
IProfile::Cursor 677
IProfileHandle 679
IProgressIndicator 681
IProgressIndicator::Style 697
IPushButton 699
IPushButton::Style 707
IRadioButton 709
IRadioButton::Style 715
IRange 717
IRectangle 719
IRefCounted 727
IReference 729
IRegionHandle 731
IResizeEvent 732
IResizeHandler 734
IResource 736
IResourceId 738
IResourceLibrary 740
IResourceLock 746
is32Bit
IProcedureAddress 671
isAcquired
ICnrEmphasisEvent 82
isAltDown
IKeyboardEvent 452
isAnExtension
IFrameWindow 361
isAttribute
IContainerObject 219
isAutoDeleteObject
IWindow 932
isAutoDestroyWindow
IWindow 932
isAutoScroll
IEntryField 263
isAutoSelect
ICheckBox 75
IRadioButton 711
ISettingButton 784
isAutoSize
INotebook::PageSettings 603
isAutoTab
IEntryField 263
isBitmap
IFont 316
isBitmapOnly
IFont 316
isBold
IFont 316
isBorder
IPushButton 702
ISpinButton 810
isCachingEnabled
IContainerControl 164
isChangeAllowed
IEditVerifyEvent 252
isChanged
IMultiLineEdit 568
isCharacter
IKeyboardEvent 452
isChecked
IMenuItem 516
isCollapsed
IContainerControl 164
isColumnExpandable
IMultiCellCanvas 550
isColumnRight
IContainerControl 165

isCommand
IEntryField 263
isComposite
IKeyboardEvent 452
IScrollBar 748
IScrollBar::Style 758
IScrollEvent 760
IScrollHandler 762
isCtrlDown
IKeyboardEvent 452
isCursored
IContainerControl 165
isCursorSelect
IRadioButton 711
isDataUpdateEnabled
IMultiLineEdit 568
isDate
IContainerColumn 142
isDefault
IPushButton 702
isDetailsView
IContainerControl 165
isDialogTemplateSet
IFileDialog::Settings 297
isDisabled
IMenuItem 516
IWindow 932
isDrawBackgroundEnabled
IContainerControl 165
isDrawItem
IListBox 466
IMenuItem 516
isDrawItemEnabled
IContainerControl 165
IProgressIndicator 686
isDropOnAble
IContainerControl 165
IContainerObject 215
ISelectHandler 766
ISemaphoreHandle 769
isEmpty
IComboBox 114
IEntryField 264
IListBox 466
INotebook 586
isEnabled
IHandler 405
IWindow 932
isEnabledForNotification
INotifier 612
isEnabledForNotification *(continued)*
IWindow 932
isEntryPoint32Bit
IDynamicLinkLibrary 247
isEqual
IContainerControl::CompareFn 199
ISetCanvas 770
ISetCanvas::Style 780
ISettingButton 783
isExpanded
IContainerControl 165
isExtendedSelect
IListBox 466
isExtendedSelection
IContainerControl 165
isFastSpin
ISpinButton 810
isFastSpinEnabled
IBaseSpinButton 36
isFixed
IFont 316
isFlashing
IFrameWindow 361
isFlowed
IContainerControl 166
isFlowedNameView
IContainerControl 166
isFlowedTextView
IContainerControl 166
isForComposite
IKeyboardEvent 452
isFrame
IHelpSubitemNotFoundEvent 420
isFramed
IMenuItem 516
isFrameWindow
IFrameWindow 373
IWindow 933
isFromFrame
ICommandEvent 131
isGroup
IControl 223
IWindow 933
isGUIInitialized
ICurrentThread 237
isHalftone
IStaticText 842
ISharedResource 788
isHeadingIconHandle
IContainerColumn 142

isHeadingReadOnly
IContainerColumn 142
isHeadingString
IContainerColumn 142
isHeadingWriteable
IContainerColumn 142
isHelp
IPushButton 702
isHighlighted
IButton 59
IMenuItem 517
isHorizontal
ICnrScrollEvent 102
IScrollBar 750
isHorizontalScroll
IComboBox 115
IListBox 466
IShowListHandler 790
isIconHandle
IContainerColumn 143
isIconView
IContainerControl 166
isInsertMode
IEntryField 264
isInUse
IContainerControl 166
IContainerObject 215
isInvalidComposite
IKeyboardEvent 452
isItalic
IFont 316
isItemChecked
IMenu 488
isItemDisabled
IMenu 488
isItemEnabled
IMenu 488
ISize 792
isLayoutDistorted
IWindow 933
isLeftDetails
ICnrScrollEvent 103
ISlider 794
ISlider::Style 803
isListShowing
IComboBox 115
isMajorTab
INotebook::PageSettings 603
isMargin
IEntryField 264
isMaster
IBaseSpinButton 36
ISpinButton 810
isMaximized
IFrameWindow 361
isMemberOf
IContainerControl::FilterFn 201
isMenu
IHelpSubitemNotFoundEvent 420
isMinimized
IFrameWindow 361
isMinorTab
INotebook::PageSettings 603
isModal
IFrameWindow 362
isModeless
IFileDialog 290
IFontDialog 327
isMoveValid
IContainerControl 167
isMultipleSelect
IListBox 466
isMultipleSelection
IContainerControl 168
isNameView
IContainerControl 168
isNoAdjustPosition
IListBox 467
isNoDismiss
IMenuItem 517
isNonPropOnly
IFont 316
isNumber
IContainerColumn 143
isOpen
IContainerObject 216
IDynamicLinkLibrary 247
IResourceLibrary 741
isOpenDialog
IFileDialog::Settings 297
isOutline
IFont 316
ISpinButton 805
ISpinButton::Cursor 819
ISpinButton::Style 821
ISpinHandler 823
ISplitCanvas 826
ISplitCanvas::Style 836
isPMInitialized
ICurrentThread 237

isPositionSet
IFileDialog::Settings 297
isPrimaryWindow
IWindow 949
isReadOnly
IContainerColumn 143
IContainerControl 168
IContainerObject 216
IEntryField 264
IMultiLineEdit 569
ISpinButton 810
isRefreshOn
IContainerControl 168
IContainerObject 216
isRelatedHandle
IFrameWindow 373
IWindow 949
isRepeat
IKeyboardEvent 452
isRibbonStripEnabled
IProgressIndicator 686
isRightDetails
ICnrScrollEvent 103
isRowExpandable
IMultiCellCanvas 550
isScanCode
IKeyboardEvent 453
isSelectable
IMenuItem 517
isSelected
IComboBox 115
IContainerControl 168
IListBox 467
ISettingButton 784
isSeparator
IMenuItem 517
isServant
IBaseSpinButton 36
ISpinButton 810
isSet
IAccelerator 26
isShiftDown
IKeyboardEvent 453
isShowing
IWindow 933
isSingleSelection
IContainerControl 168
isSizeToGraphic
IGraphicPushButton 385
isSnapToTickEnabled
IProgressIndicator 686
isSource
IContainerControl 168
isSpinFieldValid
IBaseSpinButton 36
INumericSpinButton 616
ITextSpinButton 883
isStarted
IThread 896
isStatusTextOn
INotebook::PageSettings 603
isStrikeout
IFont 316
IStaticText 842
isString
IContainerColumn 143
isSystemCommand
IPushButton 702
isTabStop
ICanvas 67
IControl 224
IWindow 933
isTarget
IContainerControl 169
IStaticText 838
IStaticText::Style 848
isTextView
IContainerControl 169
isTime
IContainerColumn 143
isTitleSeparatorVisible
IContainerControl 169
isTitleVisible
IContainerControl 169
isTitleWriteable
IContainerControl 169
isTopLevelShell
ICurrentThread 238
isTreeIconView
IContainerControl 169
isTreeNameView
IContainerControl 169
isTreeTextView
IContainerControl 170
isTreeView
IContainerControl 170
IStringHandle 851
ISubmenu 852

ISubmenu::Cursor 861
isUncombined
IKeyboardEvent 453
isUnderscore
IFont 316
IStaticText 842
isUndoable
IMultiLineEdit 569
isUpTransition
IKeyboardEvent 453
isValid
IComboBox::Cursor 125
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209
IListBox::Cursor 477
IMenu::Cursor 497
INotebook::Cursor 600
IProfile::Cursor 677
ISpinButton 810
ISpinButton::Cursor 819
ISubmenu::Cursor 861
ITextSpinButton::Cursor 888
IThread::Cursor 902
IWindow 933
IWindow::ChildCursor 956
IWindowHandle 963
isVectorOnly
IFont 317
isVertical
ICnrScrollEvent 103
IProgressIndicator 687
IScrollBar 750
isVerticalFlip
IMenu 488
isVirtual
IKeyboardEvent 453
isVisible
IContainerColumn 143
IContainerControl 170
IContainerObject 216
IWindow 933
isWindow
IHelpSubitemNotFoundEvent 420
isWordWrap
IMultiLineEdit 569
ISWP 863
ISWPArray 865
isWriteable
IBaseSpinButton 36
isWriteable *(continued)*
IContainerColumn 143
IContainerControl 170
IContainerObject 216
isXerrorCodeAvailable
ICurrentThread 238
ISystemBitmapHandle 867
ISystemMenu 869
ISystemPointerHandle 874
itemHandle
IComboBox 115
IListBox 467
itemHeight
IListBox 467
itemProvider
IWindow 933
itemRect
IMenu 489
itemText
IComboBox 115
IListBox 467
ITextControl 876
ITextSpinButton 880
ITextSpinButton::Cursor 888
ITextSpinButton::Style 890
IThread 892
IThread::Cursor 902
IThreadFn 904
IThreadId 905
IThreadMemberFn 906
ITitle 908
IViewPort 916
IViewPort::Style 924
IWindow 926
IWindow::ChildCursor 956
IWindow::ExceptionFn 958
IWindow::Style 959
IWindowHandle 962

J

justifyData
IContainerColumn 143
justifyHeading
IContainerColumn 144

K

key
IKeyboardHandler 459

keyName
ISharedResource 789
keysHelpId
IHelpHandler 412

L

last
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209
INotebook::Cursor 600
lastPage
INotebook 586
layout
ICanvas 70
IMultiCellCanvas 555
ISetCanvas 777
ISplitCanvas 833
IViewPort 922
IWindow 955
layoutAdjustment
IComboBox 115
IWindow 934
layoutSize
ICanvas 70
layoutType
IMenuItem 517, 521
left
IRectangle 722
leftCenter
IRectangle 722
leftIndex
IEntryField 264
lib files, description 5
libibmui.a 5
libibmuis.a 5
limit
IBaseSpinButton 36
IEntryField 264
IMultiLineEdit 569
IStaticText 842
lineCount
IInfoArea 444
lineDown
IScrollHandler 764
lineLeft
IScrollHandler 764
lineRight
IScrollHandler 764
lineSpacing
IContainerControl 170
lineUp
IScrollHandler 764
list box control
listShown
IShowListHandler 791
loadAccelTable
IResourceLibrary 741
loadBitmap
IResourceLibrary 741
loadDialog
IResourceLibrary 742
loadHelpTable
IResourceLibrary 742
loadIcon
IResourceLibrary 742
loadMenu
IResourceLibrary 742
loadMessage
IResourceLibrary 743
loadPointer
IResourceLibrary 743
loadString
IResourceLibrary 743
locateText
IComboBox 115
IListBox 467
location
IFrameExtension 342
IFrameWindow 376
lock
IResource 736
lostFocus
IFocusHandler 311
lowerBound
IRange 717
lowHighByte
IEventData 286
lowLowByte
IEventData 286
lowNumber
IEventData 286

M

main window
makePopUpMenu
IMenuHandler 512

mapPoint
IWindow 934
margin
ISetCanvas 772
marginSize
IGraphicPushButton 385
matchForMnemonic
ICanvas 67
IWindow 934
maxAscender
IFont 317
maxCharHeight
IFont 317
maxDescender
IFont 317
maximize
IFrameWindow 362
maximizeRect
IFrameWindow 362
maximum
IPair 652
maxLowercaseAscender
IFont 317
maxLowercaseDescender
IFont 317
maxSize
IFont 317
maxUppercaseSize
IFont 317
MBCS
member function names, conventions 5
menu bar
menuBarCommand
IHelpHandler 413
menuEnded
ICnrMenuHandler 97
IInfoArea 449
IMenuHandler 512
menuItem
IMenu 489
IMenuEvent 508
menuItemId
IMenuEvent 508
menus
menuSelected
IInfoArea 449
IMenuHandler 513
menuShowing
IMenuHandler 513
message box
messageQueue
ICurrentThread 238
IThread 896
IWindow 934
minimize
IFrameWindow 362
minimizeRect
IFrameWindow 362
minimum
IPair 652
minimumCharacters
IListBox 468
minimumRows
IComboBox 116
IListBox 468
minimumSize
IWindow 934
minScrollIncrement
IScrollBar 750
minTextWidth
IFont 317
missingText
IInfoArea 445
mixedCharacter
IKeyboardEvent 453
MLE
modelessResults
IFileDialogHandler 307
IFontDialogHandler 337
mouseAction
IMouseClickEvent 538, 539
mouseClicked
IMouseClickHandler 542
mouseNumber
IMouseClickEvent 538, 539
mousePointer
IFrameWindow 362
mousePosition
IMenuEvent 508
IMouseClickEvent 538
moveArmToPixel
IProgressIndicator 687
moveArmToTick
IProgressIndicator 687
moveBy
IRectangle 722
movedBy
IRectangle 722

movedTo
IRectangle 722
moveIconTo
IContainerControl 170
moveObjectTo
IContainerControl 171
moveScrollBox
IScrollHandler 764
moveScrollBoxTo
IScrollBar 750
moveSizeTo
IBaseSpinButton 37
IBitmapControl 49
IComboBox 116
IEntryField 264
IGraphicPushButton 385
IGroupBox 393
IHelpWindow 427
IIconControl 437
IOutlineBox 627
IProgressIndicator 687
IScrollBar 750
ISetCanvas 772
IStaticText 842
IWindow 934
moveSizeToClient
IFrameWindow 362
moveTo
IHelpWindow 427
IRectangle 722
IWindow 935
multiple-cell canvas
multiple-line edit (MLE) fields

N

name
IFont 318
IProfile 675
naming conventions
data member names 5
file names 5
function arguments 6
function return types 6
global type names 5
maximum characters 5
member function names 5
numerations 5
type names 5
national language support
newScrollBoxPosition
IScrollEvent 760
newSize
IResizeEvent 732
newStartedThread
IThread 901
next
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209
INotebook::Cursor 600
nextPage
INotebook 586
nextShellRect
IFrameWindow 363
nlsCompare
IContainerControl 171
nominalPointSize
IFontDialog 327
notebook
IPageEvent 634
IPageHelpEvent 641
IPageRemoveEvent 643
IPageSelectEvent 644
notebookSize
INotebook 586
notificationHandler
IWindow 950
notifyObservers
INotifier 612, 613
IWindow 935, 950
notifyOwner
IFrameWindow 363
number1
IEventData 287
number2
IEventData 287
numberOfApplications
IProfile 675
numberOfColumnChanges
IContainerControl 171
numberOfExtensions
IFrameHandler 352
numberOfItems
IMenu 489
numberOfKeys
IProfile 675
numberOfLines
IMultiLineEdit 569

numberOfObjectChanges
IContainerControl 171
numberOfSelections
IComboBox 116
IListBox 468
numberOfTicks
IProgressIndicator 687
numerations, conventions 5

O

object
ICnrEmphasisEvent 83
ICnrEnterEvent 85
ICnrHelpEvent 93
objectAt
IContainerControl 172
objectCopy
IContainerObject 216
objectCount
IContainerControl 172
objectList
IContainerControl 172
objectText
ITitle 910
objectUnderPoint
IContainerControl 172
objectWindow
IWindow 935
observerList
INotifier 613
IWindow 950
OKButtonText
IFileDialog::Settings 297
oldSize
IResizeEvent 732
open
IDynamicLinkLibrary 247
openLibrary
IHelpHandler 413
operator -
IPair 654
operator -=
IPair 654
operator ->
IReference 730
operator _WidgetRec *
IWindowHandle 963
operator _XDisplay *
IDisplayHandle 245
operator _XGC *
IPresSpaceHandle 666
operator _XrmHashBucketRec *
IProfileHandle 680
operator _XtAppStruct *
IContextHandle 221
operator !=
IBitFlag 45
IColor 107
IPair 652
IRectangle 723
operator /
IPair 654
operator /=
IPair 655
operator []
ISWPArray 865
operator *
IPair 653
IReference 730
operator *=
IPair 654
operator &
IRectangle 723
operator &=
IRectangle 723
operator %
IPair 652
operator %=
IPair 653
operator +
IPair 654
operator +=
IPair 654
operator <
IPair 655
operator <<
IPair 655
IRectangle 723
operator <=
IPair 655
operator =
IBitmapHandle 55
IColor 107
IDynamicLinkLibrary 248
IFont 318
IFontDialog::Settings 331
IPointerHandle 659
IProfile 675
IReference 730

operator = *(continued)*
IResourceLibrary 744
IThread 901
operator ==
IBitFlag 45
IColor 108
IContainerControl 172
IContainerObject 216
IPair 655
IRectangle 723
operator >
IPair 656
operator >=
IPair 656
operator |
IRectangle 723
operator |=
IRectangle 723
operator char *
IEventData 287
operator delete
IContainerObject 217
operator new
IContainerObject 217
operator PtrToFnType
IProcedureAddress 671
operator T *
IReference 730
operator unsigned long
IEventData 287
IResourceId 739
IThreadId 905
operator Value
IAccelTblHandle 28
IAnchorBlockHandle 29
IBitmapHandle 56
IEnumHandle 276
IMessageQueueHandle 534
IModuleHandle 536
IPageHandle 636
IPointerHandle 659
IPresSpaceHandle 666
IProcessId 672
IProfileHandle 679
IRegionHandle 731
ISemaphoreHandle 769
IStringHandle 851
ISystemBitmapHandle 867
ISystemPointerHandle 874
IThreadId 905
operator Value *(continued)*
IWindowHandle 963
orientation
INotebook 586, 597
ISplitCanvas 829, 834
origDefaultButtonHandle
ICanvas 67
outlineType
IOutlineBox 627, 631
owner
IAccelerator 26
IWindow 935

P

packType
ISetCanvas 772, 779
pad
ISetCanvas 772
pageDown
IScrollHandler 764
pageHandle
IPageEvent 634
IPageHelpEvent 641
IPageSelectEvent 644
pageLeft
IScrollHandler 764
pageRight
IScrollHandler 764
pageScrollIncrement
IScrollBar 750
pageSettings
INotebook 586
pageSize
INotebook 587
pagesToEnd
INotebook 587
pagesToMajorTab
INotebook 587
pagesToMinorTab
INotebook 587
pageUp
IScrollHandler 765
pageWindow
IPageRemoveEvent 643
paintWindow
IPaintHandler 650
parameter1
IEvent 281

parameter2
IEvent 281
parent
IWindow 936
parentObject
IContainerControl 172
passEventToOwner
ICanvas 70
IEntryField 269
IListBox 473
IMultiLineEdit 576
IPushButton 705
IScrollBar 756
ISettingButton 786
IStaticText 846
IWindow 950
passToOwner
IEvent 281
paste
IEntryField 265
IMultiLineEdit 569
pib
ICurrentApplication 235
pointSize
IFont 318
IFontDialog 327
pointSizeAt
IFont 318
pop-up menu
popupMenuObject
ICnrMenuHandler 96
portability
position
IComboBox 116
IFileDialog::Settings 298
IGroupBox 394
IOutlineBox 627
IScrollBar 750
ISetCanvas 773
ISWP 863
IWindow 936
positionBehindSibling
IWindow 936
positionBehindSiblings
IWindow 936
positionExtensions
IFrameHandler 353
positionOnSiblings
IWindow 936
positionViewWindow
IViewPort 922
postEvent
IMessageQueueHandle 534
IWindow 937
IWindowHandle 963
postEvents
IMessageQueueHandle 535
IWindowHandle 963
prepareForUse
IWindow 950
Presentation Manager messages
WM_CONTEXTMENU 510
WM_INITMENU 508, 510
WM_MENUEND 508, 510
WM_MENUSELECT 508, 510
pressedOK
IFileDialog 290
IFontDialog 327
presSpace
IWindow 937
presSpaceHandle
IPaintEvent 648
previous
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209
INotebook::Cursor 600
previousPage
INotebook 587
previousSelectedPageHandle
IPageSelectEvent 645
prevScrollBoxPosition
IScrollBar 751
primaryScale
IProgressIndicator 687
PriorityClass
IApplication 32
IThread 896
priorityLevel
IThread 896
procAddress
IDynamicLinkLibrary 248
processMsgs
ICurrentThread 238
proposedText
IEditVerifyEvent 252
pull-down menu
push button control

Q

queueSize
IThread 896

R

radio button control
range
INumericSpinButton 616
ISpinButton 810
rect
IGroupBox 394
IPaintEvent 648
IWindow 937
redMix
IColor 108
refresh
IContainerControl 173
IContainerObject 217
IWindow 937
refreshAllContainers
IContainerControl 173
refreshTabs
INotebook 588
refreshText
ISpinButton 811
registerCallbacks
IBaseSpinButton 40
ICanvas 70
ICheckBox 77
IComboBox 122
IContainerControl 189
IEntryField 269
IFileDialog 294
IFrameWindow 374
IListBox 473
IMenuBar 504
IMultiLineEdit 576
INotebook 596
INumericSpinButton 620
IPopUpMenu 665
IProgressIndicator 693
IPushButton 705
IRadioButton 714
IScrollBar 756
ISpinButton 817
IWindow 950
related books 8
relatedHandlesList
relatedHandlesList *(continued)*
IThread 897
relativeSize
IFrameExtension 342
releasePresSpace
IWindow 938
remainingStack
ICurrentThread 238
remove
IAccelerator 26
IComboBox 116
IListBox 468
IPageHandler 639
removeAll
IComboBox 117
IEntryField 265
IListBox 468
IMultiLineEdit 570
ISpinButton 811
ITextSpinButton 883
removeAllObjects
IContainerControl 173
removeAllObservers
INotifier 613
IWindow 951
removeAllPages
INotebook 588
removeAt
IComboBox 117
IListBox 468
ISpinButton 811
ITextSpinButton 883
removeBorder
IBaseSpinButton 37
IPushButton 702
removeChangedFlag
IMultiLineEdit 570
removeColumn
IContainerControl 173
removeColumnAt
IContainerControl 173
removeConditionalCascade
IMenu 489
removeDefaultHandler
IFrameWindow 374
removeDetent
ISlider 798
removeExtension
IFrameWindow 363

removeFromCell
IMultiCellCanvas 550
removeFromWindowList
IFrameWindow 363
removeFromWindowSet
IWindow 951
removeHandler
IWindow 951
removeInUse
IContainerControl 173
IContainerObject 217
removeLine
IMultiLineEdit 570
removeObject
IContainerControl 174
removeObjectAt
IContainerControl 174
removeObserver
INotifier 613
IWindow 951
removePage
INotebook 588
removeRef
IRefCounted 728
removeRelatedHandleFromWindowSet
IWindow 951
removeSelected
IContainerControl 174
removeSelectedObjects
IContainerControl 174
removeSourceEmphasis
ICnrMenuHandler 97
IMenuHandler 513
removeSubmenu
IMenu 490
ISubmenu 856
removeSubmenuAt
IMenu 490
ISubmenu 856
removeTabSection
INotebook 588
repeatCount
IKeyboardEvent 453
replaceAt
IComboBox 117
IListBox 468
ISpinButton 811
ITextSpinButton 883
requirements
reset
IAccelerator 26
resetActiveColor
IWindow 938
resetActiveTextBackgroundColor
ITitle 910
resetActiveTextForegroundColor
ITitle 910
resetBackgroundColor
IMenu 490
IWindow 938
resetBackgroundMajorColor
INotebook 588
resetBackgroundMinorColor
INotebook 588
resetBackgroundPageColor
INotebook 588
resetBorderColor
IWindow 938
resetChangedFlag
IMultiLineEdit 570
resetColor
IWindow 951
resetDisabledBackgroundColor
IMenu 490
IWindow 938
resetDisabledForegroundColor
IMenu 490
IWindow 939
resetFillColor
IStaticText 842
resetForegroundColor
IMenu 490
IScrollBar 751
IWindow 939
resetForegroundMajorColor
INotebook 589
resetForegroundMinorColor
INotebook 589
resetHiliteBackgroundColor
IMenu 490
IWindow 939
resetHiliteForegroundColor
IMenu 491
IScrollBar 751
IWindow 939
resetInactiveColor
IWindow 939
resetInactiveTextBackgroundColor
ITitle 910

resetInactiveTextForegroundColor
ITitle 911
resetMinimumSize
IWindow 939
resetShadowColor
IWindow 939
resetSplitBarEdgeColor
ISplitCanvas 829
resetSplitBarMiddleColor
ISplitCanvas 829
resize
IPageHandler 639
resourceLibrary
ICurrentApplication 234
IInfoArea 445
IResourceId 739
resources
Response
IMessageBox 529
restore
IFrameWindow 364
restoreRect
IFrameWindow 364
result
IEvent 281
IFrameWindow 364
resultingText
IEditVerifyEvent 253
resume
IThread 897
returnValue
IFileDialog 291
IFontDialog 327
right
IRectangle 723
rightCenter
IRectangle 723
rowHeight
IMultiCellCanvas 551
run
ICurrentApplication 234
IThreadFn 904
IThreadMemberFn 906

S

sample
sample directory location 6
saveAsEAType
IFileDialog 291
saved
IFrameHandler 352
Scale
IProgressIndicator 695
scaleBy
IPair 656
IRectangle 724
scaledBy
IPair 656
IRectangle 724
scanCode
IKeyboardEvent 453
scanCodeKeyPress
IKeyboardHandler 459
scroll
IContainerControl 174
scrollableRange
IScrollBar 751
scrollAmount
IScrollEvent 760
scrollBarWindow
IScrollEvent 760
scrollBoxPosition
IScrollBar 751
scrollBoxRange
IScrollBar 751
scrollBoxTrack
IScrollHandler 765
scrollBoxTrackEnd
IScrollHandler 765
scrollDetailsHorizontally
IContainerControl 174
scrollEnd
IScrollHandler 765
scrollHorizontally
IContainerControl 176
scrollToObject
IContainerControl 176
scrollVertically
IContainerControl 176
scrollViewHorizontallyTo
IViewPort 918
scrollViewVerticallyTo
IViewPort 918
searchListWindow
IHelpWindow 427
SearchType
IComboBox 124
IListBox 474

select
IComboBox 117
IListBox 469
IPageHandler 639
ISettingButton 784
selectAll
IListBox 469
selected
ISelectHandler 768
selectedChanged
ICnrHandler 91
selectedFileCount
IFileDialog 291
selectedIndex
IRadioButton 711
selectedRange
IEntryField 265
IMultiLineEdit 570
selectedText
IEntryField 265
IMultiLineEdit 570
selectedTextLength
IMultiLineEdit 570
selection
IComboBox 117
IListBox 469
selectItem
IMenu 491
selectRange
IEntryField 265
IMultiLineEdit 571
sendEvent
IContainerControl 176
IHelpWindow 427
IWindow 940
IWindowHandle 963
sendEvents
IWindowHandle 964
separatorHandle
IFrameExtension 342
separatorType
IFrameExtension 343
IFrameWindow 376
separatorWidth
IFrameExtension 343
set
IAccelerator 26
set canvas
setAccelerator
IHelpWindow::Settings 433
setAccelResLibrary
IHelpWindow::Settings 433
setActiveColor
IWindow 940
setActiveTextBackgroundColor
ITitle 911
setActiveTextForegroundColor
ITitle 911
setActiveWindow
IHelpWindow 427
setAlignment
IBaseSpinButton 37
IEntryField 265
ISetCanvas 773
ISpinButton 811
IStaticText 843
setAllEmphasis
IFont 318
setArgs
ICurrentApplication 234
setArmSize
ISlider 798
setAssociatedWindow
IHelpWindow 428
setAttribute
IMenuItem 517
setAttributes
IContainerControl 189
IContainerObject 219
setAutoDeleteObject
IWindow 940
setAutoDestroyWindow
IWindow 940
setAutoInitGUI
IThread 897
setAutoInitPM
IThread 897
setBackgroundColor
IComboBox 118
IContainerControl 177
IMenu 491
INotebook 589
IProgressIndicator 688
IWindow 940
setBackgroundMajorColor
INotebook 589
setBackgroundMinorColor
INotebook 589
setBackgroundPageColor
INotebook 589

setBase
IContainerObject 219
setBinding
INotebook 589
setBitmap
IBitmapControl 49
IMenu 491
IMenuItem 518
ISubmenu 856
setBlue
IColor 108
setBold
IFont 318
setBorderColor
IWindow 941
setBorderHeight
IFrameWindow 364
setBorderSize
IFrameWindow 364
setBorderWidth
IFrameWindow 365
setChangedFlag
IEntryField 266
IMultiLineEdit 571
setCharHeight
IFont 318
setCharSize
IFont 319
setCharType
IEntryField 266
setCharWidth
IFont 319
setChecked
IMenuItem 518
setClient
IFrameWindow 365
setClientRect
IFrameFormatEvent 348
setClosed
IContainerObject 217
setColor
color support 965
IWindow 951
setColumnInfo
IContainerColumn 147
setColumnWidth
IMultiCellCanvas 551
setCommand
IMenuItem 518
setConditionalCascade
IMenu 491
setContainer
IContainerColumn 147
setContainerAttributes
IContainerControl 190
setControlHandle
IEvent 281
setCoord1
IPair 656
setCoord2
IPair 656
setCurrent
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 204
IContainerControl::TextCursor 209
INotebook::Cursor 600
ISpinButton 812
setCursor
IContainerControl 177
setCursorAt
IMultiLineEdit 571
setCursorAtLine
IMultiLineEdit 571
setDataAttributes
IContainerColumn 147
setDataOffset
IContainerColumn 144
setDeckCount
ISetCanvas 773
setDeckOrientation
ISetCanvas 773
setDefaultApplicationName
IProfile 675
setDefaultAttribute
IContainerControl 177
IMenuItem 518
setDefaultAutoInitGUI
IThread 897
setDefaultAutoInitPM
IThread 898
setDefaultCell
IMultiCellCanvas 552
setDefaultContainer
IContainerControl 190
setDefaultDataStyle
IContainerColumn 144
setDefaultHeadingStyle
IContainerColumn 144

setDefaultOrdering
IFrameWindow 365
IWindow 941
setDefaultQueueSize
IThread 898
setDefaultStackSize
IThread 898
setDefaultStyle
IBitmapControl 49
ICanvas 67
ICheckBox 75
IComboBox 118
IContainerControl 177
IEntryField 266
IFileDialog 291
IFontDialog 327
IFrameWindow 365
IGraphicPushButton 385
IGroupBox 394
IIconControl 437
IListBox 469
IMenu 492
IMenuBar 502
IMenuItem 518
IMultiCellCanvas 552
IMultiLineEdit 572
INotebook 589
INumericSpinButton 616
IOutlineBox 628
IProgressIndicator 688
IPushButton 702
IRadioButton 711
IScrollBar 751
ISetCanvas 773
ISlider 798
ISpinButton 812
ISplitCanvas 829
IStaticText 843
ITextSpinButton 883
IViewPort 919
setDeleteColumnsOnClose
IContainerControl 177
setDeleteObjectsOnClose
IContainerControl 178
setDestroyOnClose
IFrameWindow 366
setDetailsViewSplit
IContainerControl 178
setDialogTemplate
IFileDialog::Settings 298
setDialogTemplate *(continued)*
IFontDialog::Settings 332
setDirection
IFont 319
setDisabled
IMenuItem 518
setDisabledBackgroundColor
IMenu 492
IWindow 941
setDisabledForegroundColor
IMenu 492
IWindow 941
setDisabledText
IInfoArea 445
setDispatchingHandle
IEvent 282
setDisplayPS
IFontDialog::Settings 332
setDisplayWidth
IContainerColumn 144
setDrawItem
IMenuItem 519
setEditColumn
IContainerControl 178
setEditMLE
IContainerControl 178
setEditObject
IContainerControl 178
setEditRegion
IMultiLineEdit 572
setEditRegionHeight
IMultiLineEdit 572
setEditRegionWidth
IMultiLineEdit 572
setEmphasis
IContainerControl 190
IContainerObject 219
setEndPosition
IEditVerifyEvent 253
setEventType
IEvent 282
setExceptionFunction
IWindow 941
setExtendedSelection
IContainerControl 179
setExtensions
IFrameWindow 374
setExtensionSize
IFrameWindow 366

setFamily
IFontDialog::Settings 332
setFastSpin
ISpinButton 812
setFileName
IFileDialog::Settings 298
setFillColor
IStaticText 843
setFocus
IWindow 941
setFont
IMultiLineEdit 572
IWindow 942
setFontAngle
IFont 319
setFontShear
IFont 319
setForegroundColor
IComboBox 118
IMenu 492
INotebook 590
IOutlineBox 628
IProgressIndicator 688
IScrollBar 752
IWindow 942
setForegroundMajorColor
INotebook 590
setForegroundMinorColor
INotebook 590
setFramed
IMenuItem 519
setGraphic
IGraphicPushButton 385
setGraphicContext
IPaintEvent 648
setGreen
IColor 108
setGroupPad
ISetCanvas 773
setHandle
IEvent 282
setHeadingIcon
IContainerColumn 145
setHeadingText
IContainerColumn 145
setHeight
IListBox 470
ISize 793
setHelpId
IContainerColumn 145
setHelpResLibrary
IHelpWindow::Settings 433
setHelpTable
IHelpWindow 428
IHelpWindow::Settings 434
setHighlighted
IMenuItem 519
setHiliteBackgroundColor
IMenu 492
IWindow 942
setHiliteForegroundColor
IMenu 492
IScrollBar 752
IWindow 942
setHomePosition
IProgressIndicator 688
setIcon
IContainerObject 217
IFrameWindow 366
IIconControl 437
setIconSize
IContainerControl 179
setIconText
IContainerObject 217
setId
IApplication 32
IWindow 942
setInactiveColor
IWindow 942
setInactiveText
IInfoArea 445
setInactiveTextBackgroundColor
ITitle 911
setInactiveTextForegroundColor
ITitle 911
setIndex
IMenuItem 519
setInitialDrive
IFileDialog::Settings 298
setInitialFileType
IFileDialog::Settings 298
setInputType
ISpinButton 812
setInUse
IContainerControl 179
IContainerObject 218
setItalic
IFont 320
setItem
IMenu 493

setItem *(continued)*
ISubmenu 856
setItemHandle
IComboBox 118
IListBox 470
setItemHeight
IListBox 470
setItemProvider
IWindow 943
setItemText
IComboBox 118
IListBox 470
setLayout
IMenuItem 519
setLayoutDistorted
ICanvas 67
IComboBox 122
IEntryField 269
IFrameWindow 366
IMultiCellCanvas 552
ISetCanvas 774
ISplitCanvas 829
ITextControl 877
IViewPort 919
IWindow 943
setLayoutSize
ICanvas 71
setLeftIndex
IEntryField 266
setLibraries
IHelpWindow::Settings 434
setLimit
IBaseSpinButton 37
IEntryField 266
IMultiLineEdit 572
INumericSpinButton 616
ISpinButton 813
IStaticText 843
ITextSpinButton 884
setLineCount
IInfoArea 445
setLineSpacing
IContainerControl 179
setLock
IResourceLock 747
setLowerBound
IRange 717
setMajorTabSize
INotebook 590
setMargin
ISetCanvas 774
setMarginSize
IGraphicPushButton 386
setMaster
IBaseSpinButton 37
ISpinButton 813
setMenu
IMenuBar 502
setMenuBar
IHelpWindow::Settings 434
setMinimumCharacters
IListBox 470
setMinimumRows
IComboBox 118
IListBox 470
setMinimumSize
IWindow 943
setMinorTabSize
INotebook 590
setMinScrollIncrement
IScrollBar 752
setMissingText
IInfoArea 445
setMixedTargetEmphasis
IContainerControl 179
setMousePointer
IFrameWindow 367
setMultipleSelection
IContainerControl 179
setName
IFont 320
setNoDismiss
IMenuItem 519
setNormalTargetEmphasis
IContainerControl 179
setNotificationHandler
IWindow 952
setObjectText
ITitle 912
setOKButtonText
IFileDialog::Settings 299
setOpen
IContainerObject 218
setOpenDialog
IFileDialog::Settings 299
setOrderedTargetEmphasis
IContainerControl 180
setOrientation
INotebook 590

setOrientation *(continued)*
ISplitCanvas 830
setOutline
IFont 320
setOutlineType
IOutlineBox 628
setOwner
IWindow 943
setPackType
ISetCanvas 774
setPad
ISetCanvas 774
setPageButtonSize
INotebook 591
setPageScrollIncrement
IScrollBar 752
setParent
IWindow 943
setPassToOwner
IEvent 282
setPointSize
IFont 320
IFontDialog::Settings 332
setPosition
IFileDialog::Settings 299
IFontDialog::Settings 332
ISWP 863
setPreviewText
IFontDialog::Settings 332
setPrevScrollBoxPosition
IScrollBar 753
setPrimaryScale
IProgressIndicator 688
setPrinterPS
IFontDialog::Settings 332
setPriority
IApplication 31
IThread 898
setProposedText
IEditVerifyEvent 253
setQueueSize
IThread 898
setRange
INumericSpinButton 616
ISpinButton 813
setReadOnly
ISpinButton 813
setRed
IColor 108
setRefreshOff
IContainerControl 180
IContainerObject 218
setRefreshOn
IContainerControl 180
IContainerObject 218
setRelatedHandlesList
IThread 899
setResourceLibrary
ICurrentApplication 234
IInfoArea 446
setRestoreRect
IFrameWindow 367
setResult
IEvent 283
IFrameWindow 367
setRowHeight
IMultiCellCanvas 552
setSaveAsDialog
IFileDialog::Settings 299
setScrollableRange
IScrollBar 753
setScrollBar
IScrollBar 753
setSelectable
IMenuItem 520
setSelected
IContainerControl 180
setSeparator
IMenuItem 520
setSeparatorHandle
IFrameExtension 343
setShadowColor
IWindow 944
setShaftBreadth
IProgressIndicator 688
setShaftPosition
IProgressIndicator 689
setSingleSelection
IContainerControl 180
setSize
IFrameExtension 343
ISWP 864
setSizeList
IFontDialog::Settings 333
setSplitBarEdgeColor
ISplitCanvas 830
setSplitBarMiddleColor
ISplitCanvas 830

setSplitBarThickness
ISplitCanvas 830
setSplitWindowPercentage
ISplitCanvas 830
setStackSize
IThread 899
setStartPosition
IEditVerifyEvent 253
setStatusText
INotebook 591
INotebook::PageSettings 603
setStatusTextAlignment
INotebook 591
setStrikeout
IFont 320
setStringTableOffset
IInfoArea 446
setStyle
IEntryField 270
IMenuItem 520
IWindow 952
setSubmenu
IMenu 493
setSubmenuHandle
IMenuItem 520
setTab
IMultiLineEdit 573
setTabBitmap
INotebook 591
INotebook::PageSettings 604
setTabShape
INotebook 591
setTabText
INotebook 592
INotebook::PageSettings 604
setTabTextAlignment
INotebook 592
setText
IButton 59
IGroupBox 394
IMenu 493
IMenuItem 520
IMultiLineEdit 573
ISetCanvas 774
ISpinButton 813
IStaticText 844
ISubmenu 857
ITextControl 877
ITextSpinButton 884
ITitle 912
setTickLength
IProgressIndicator 689
setTicks
IProgressIndicator 689
setTickText
IProgressIndicator 690
Setting classes
hierarchy 18
settings
overview 3
settings class
overview 3
setTitle
IContainerControl 180
IFileDialog::Settings 299
IFontDialog::Settings 333
IHelpWindow 428
IHelpWindow::Settings 434
IMessageBox 527
setTitleAlignment
IContainerControl 181
setTitleAttributes
IContainerColumn 147
setTitleText
ITitle 912
setToFirst
IComboBox::Cursor 126
IContainerControl::ColumnCursor 197
IContainerControl::ObjectCursor 205
IContainerControl::TextCursor 210
IListBox::Cursor 477
IMenu::Cursor 497
INotebook::Cursor 600
IProfile::Cursor 678
ISpinButton::Cursor 819
ISubmenu::Cursor 861
ITextSpinButton::Cursor 888
IThread::Cursor 903
IWindow::ChildCursor 956
setToIndex
IComboBox::Cursor 126
IListBox::Cursor 477
setToLast
IComboBox::Cursor 126
IContainerControl::ColumnCursor 198
IContainerControl::ObjectCursor 205
IContainerControl::TextCursor 210
IListBox::Cursor 477
IMenu::Cursor 497
INotebook::Cursor 600

setToLast *(continued)*
IProfile::Cursor 678
ISpinButton::Cursor 819
ISubmenu::Cursor 861
ITextSpinButton::Cursor 888
setToNext
IComboBox::Cursor 126
IContainerControl::ColumnCursor 198
IContainerControl::ObjectCursor 205
IContainerControl::TextCursor 210
IListBox::Cursor 477
IMenu::Cursor 497
INotebook::Cursor 601
IProfile::Cursor 678
ISpinButton::Cursor 820
ISubmenu::Cursor 861
ITextSpinButton::Cursor 889
IThread::Cursor 903
IWindow::ChildCursor 956
setToolBarList
IFrameWindow 367
setTop
IComboBox 119
IListBox 470
IMultiLineEdit 573
setTopLeftViewPoint
IViewPort 922
setTopLevelShell
ICurrentThread 239
setToPrevious
IComboBox::Cursor 126
IContainerControl::ColumnCursor 198
IContainerControl::ObjectCursor 205
IContainerControl::TextCursor 210
IListBox::Cursor 477
IMenu::Cursor 498
INotebook::Cursor 601
IProfile::Cursor 678
ISpinButton::Cursor 820
ISubmenu::Cursor 862
ITextSpinButton::Cursor 889
setTreeExpandIconSize
IContainerControl 181
setTreeItemIcons
IContainerControl 181
setTreeViewIndent
IContainerControl 181
setTutorial
IHelpWindow::Settings 434
setUnderscore
IFont 320
setUpperBound
IRange 718
setupScrollBars
IViewPort 922
setUserData
INotebook 592
INotebook::PageSettings 604
setUserResourceLibrary
ICurrentApplication 234
setUsingHelp
IHelpWindow 428
IHelpWindow::Settings 434
setValue
IBitFlag 46
INumericSpinButton 617
ISpinButton 814
setVariable
IThread 899
setVerticalFlip
IMenu 493
setViewNum
ITitle 912
setViewText
ITitle 912
setViewWindowSize
IViewPort 919
setVisibleCount
IScrollBar 753
setWidth
ISize 793
setWindow
INotebook 592
setWindowData
IWindow 952
setWindowFont
IFont 320
setWindowList
IThread 899
setX
IPoint 658
setXerrorCode
ICurrentThread 239
setY
IPoint 658
Severity
IMessageBox 529
shadowColor
IWindow 944

shaftPosition
IProgressIndicator 690
shaftSize
IProgressIndicator 690
shareParentDBCSStatus
IFrameWindow 367
show
IContainerColumn 146
IContainerObject 218
IFrameWindow 368
IHelpWindow 428
IListBox 471
IMessageBox 527
IPopUpMenu 662
IWindow 944
showContents
IHelpHandler 413
showCoverPage
IHelpHandler 413
showDetailsView
IContainerControl 181
showDetailsViewTitles
IContainerControl 181
showFlowedNameView
IContainerControl 182
showFlowedTextView
IContainerControl 182
showHistory
IHelpHandler 413
showIconView
IContainerControl 182
showIndex
IHelpHandler 413
showList
IComboBox 119
showModally
IFrameWindow 368
showNameView
IContainerControl 182
showObject
IContainerControl 182
showPage
IHelpHandler 413
showPanelIds
IHelpWindow 429
showSearchList
IHelpHandler 414
showSeparators
IContainerColumn 146
showSourceEmphasis
IContainerControl 182
IWindow 944
showSplitBar
IContainerControl 183
showTextView
IContainerControl 183
showTitle
IContainerControl 184
showTitleSeparator
IContainerControl 184
showTreeIconView
IContainerControl 184
showTreeLine
IContainerControl 184
showTreeNameView
IContainerControl 184
showTreeTextView
IContainerControl 184
showTutorial
IHelpHandler 414
shrinkBy
IRectangle 724
shrunkBy
IRectangle 724
SiblingOrder
IWindow 955
size
IComboBox 119
IGroupBox 395
IOutlineBox 628
IRectangle 724
IScrollBar 753
ISetCanvas 774
ISWP 864
ISWPArray 866
IWindow 944
sizeBitmapTo
IResourceLibrary 744
sizeBy
IRectangle 725
sizedBy
IRectangle 725
sizedTo
IRectangle 725
sizeTo
IFrameExtension 343
IHelpWindow 429
IRectangle 725
IWindow 944

sleep
ICurrentThread 239
slider control
sort
IContainerControl 184
sortByIconText
IContainerControl 185
source
ICommandEvent 131, 132
spin button control
spinDown
IBaseSpinButton 37
INumericSpinButton 617
ISpinButton 814
ITextSpinButton 884
spinEnded
ISpinHandler 825
spinTo
INumericSpinButton 617
ISpinButton 814
ITextSpinButton 884
spinUp
IBaseSpinButton 38
INumericSpinButton 617
ISpinButton 814
ITextSpinButton 885
split canvas
SplitBarArea
ISplitCanvas 835
splitBarEdgeColor
ISplitCanvas 831
splitBarMiddleColor
ISplitCanvas 831
splitBarOffset
IContainerControl 185
splitBarThickness
ISplitCanvas 831
splitWindowPercentage
ISplitCanvas 831
stackSize
IThread 899
start
IFrameWindow 374
IThread 899
startedThread
ICurrentThread 242
IThread 901
startHandlingEventsFor
IWindow 953
startPosition
IEditVerifyEvent 253
static text control
status area
status line
statusText
INotebook::PageSettings 605
statusTextAlignment
INotebook 592
stop
IThread 900
stopHandlingEventsFor
ICnrHandler 91
ICnrMenuHandler 96
IFileDialogHandler 306
IHandler 405
IHelpHandler 411
IInfoArea 446
IMenuHandler 511
stopProcessingMsgs
IThread 900
stringTableOffset
IInfoArea 446
Stroustrup, Bjarne 9
Style
IContainerColumn 150
IMenuItem 520
IWindow 953
style classes
hierarchy 18
overview 3
styleFrom
IFrameWindow 374
styles
subitemNotFound
IHelpHandler 414
submenuHandle
IMenuItem 520
subscriptOffset
IFont 321
subscriptSize
IFont 321
subtopicId
IHelpSubitemNotFoundEvent 420
superscriptOffset
IFont 321
superscriptSize
IFont 321
support classes
hierarchy 19

support classes *(continued)*
overview 3
suspend
ICurrentThread 239
IThread 900
swapPage
IHelpHandler 414
swpArray
IFrameFormatEvent 348
sysColor
IGUIColor 399, 400
system menu
systemCommand
ICommandHandler 136
systemProfile
IProfile 676
systemScrollBarWidth
IScrollBar 753
systemScrollBoxLength
IScrollBar 754
systemScrollButtonLength
IScrollBar 754

T

tabBitmap
INotebook::PageSettings 605
IPageRemoveEvent 643
tabShape
INotebook 593, 598
tabText
INotebook::PageSettings 605
tabTextAlignment
INotebook 593
task and samples cross-reference 975
terminateGUI
ICurrentThread 239
terminatePM
ICurrentThread 240
text
IEntryField 267
IMenuItem 520
IMultiLineEdit 573
ISetCanvas 775
ISpinButton 814
ITextControl 877
ITextSpinButton 885
ITitle 912
text string
text widgets 258, 559
TextAlignment
INotebook 598
textLength
IMultiLineEdit 574
ITextControl 877
ITitle 913
textLines
IFont 321
textRectangle
IContainerControl 185
textWidth
IFont 321
threadId
IThread::Cursor 903
tickLength
IProgressIndicator 690
tickPosition
IProgressIndicator 691
tickSpacing
IProgressIndicator 691
tickText
IProgressIndicator 691
title
IContainerControl 185
IFileDialog::Settings 299
TitleAlignment
IContainerControl 192
titleAttributes
IContainerColumn 148
titleRectangle
IContainerControl 185
toolBarList
IFrameWindow 368
tools
top
IComboBox 119
IListBox 471
IMultiLineEdit 574
IRectangle 725
topCenter
IRectangle 725
topHandle
IBaseSpinButton 38
IComboBox 119
IContainerControl 190
IFrameWindow 368
IListBox 471
IMultiLineEdit 574
IPopUpMenu 662

topHandle *(continued)*
ISpinButton 814
IWindow 945
topicId
IHelpSubitemNotFoundEvent 421
topLeft
IRectangle 726
topLeftViewPoint
IViewPort 923
topPage
INotebook 593
topRight
IRectangle 726
totalPages
INotebook 593
totalRectFor
IFrameExtension 344
transpose
IPair 656
treeCollapsed
ICnrHandler 91
treeExpanded
ICnrHandler 91
tryToLoadBitmap
IResourceLibrary 744
tryToLoadDialog
IFrameWindow 375
tryToLoadIcon
IResourceLibrary 744
tryToLoadMessage
IResourceLibrary 744
tryToLoadString
IResourceLibrary 744
turnToPage
INotebook 593
tutorialName
IHelpTutorialEvent 422
type
IComboBox 119
IFrameExtension 344
ISpinButton 818
type names, conventions 5

U

uncheckItem
IMenu 494
ISubmenu 857
undo
IMultiLineEdit 574
undo *(continued)*
ISubmenu 857
unhighlight
IButton 59
unlock
IResource 736
unregisterCallbacks
IBaseSpinButton 40
ICanvas 71
ICheckBox 78
IComboBox 122
IContainerControl 190
IEntryField 270
IFileDialog 294
IFrameWindow 375
IListBox 473
IMenuBar 505
IMultiLineEdit 577
INotebook 596
INumericSpinButton 620
IPopUpMenu 665
IProgressIndicator 694
IPushButton 705
IRadioButton 714
IScrollBar 756
ISpinButton 817
IWindow 953
update
IFrameWindow 368
upperBound
IRange 718
useBitmapOnly
IFont 322
useCount
IContainerObject 220
IRefCounted 728
useExtensionMinimumSize
IFrameWindow 368
useMinimumSize
IFrameExtension 344
useNonPropOnly
IFont 322
User Interface Class Library
description 2
userData
INotebook::PageSettings 605
userProfile
IProfile 676
userResourceLibrary
ICurrentApplication 235

usesDialogBackground
IFrameWindow 368
useVectorOnly
IFont 322

V

validate
IFileDialogHandler 307
IRectangle 726
validateName
IFileDialogHandler 307
validObject
ICnrEnterEvent 86
value
IColor 108
INumericSpinButton 617
ISpinButton 815
variable
IThread 901
VerticalAlignment
IContainerColumn 148
verticalDataAlignment
IContainerColumn 146
verticalHeadingAlignment
IContainerColumn 146
verticalScrollBar
IViewPort 919
viewedPagesWindow
IHelpWindow 429
viewNum
ITitle 913
viewport
viewPortOnWindow
IContainerControl 185
viewPortOnWorkspace
IContainerControl 185
viewText
ITitle 913
viewWindow
IViewPort 919
viewWindowDrawRectangle
IViewPort 919
viewWindowSize
IViewPort 920
virtualKey
IKeyboardEvent 454
virtualKeyPress
IKeyboardHandler 459
visibleCount
IScrollBar 754
visibleLines
IMultiLineEdit 574
visibleRectangle
IComboBox 119
IWindow 945

W

waitFor
ICurrentThread 240
waitForAllThreads
ICurrentThread 240
waitForAnyThread
ICurrentThread 240
width
IRectangle 726
ISize 793
willDeleteColumnsOnClose
IContainerControl 186
willDeleteObjectsOnClose
IContainerControl 186
willDestroyOnClose
IFrameWindow 369
window
IEvent 283
INotebook 593
window classes
hierarchy 20
overview 3
windowHandle
IHelpNotifyEvent 419
ISWP 864
windowInCell
IMultiCellCanvas 553
windowList
IThread 901
windowResize
IResizeHandler 735
windows
windowScrolled
ICnrHandler 91
windowSystemClass
IFrameWindow 375
windowULong
IWindow 954
windowUShort
IWindow 954

windowWithHandle
IWindow 945
windowWithOwner
IWindow 945
windowWithParent
IWindow 945
WM_COMMAND 130, 133
WM_CONTEXTMENU 510
WM_CONTROL 229
WM_INITMENU 508, 510
WM_MENUEND 508, 510
WM_MENUSELECT 508, 510
WM_SYSCOMMAND 130

X

x
IPoint 658
X/Motif messages
WM_CONTEXTMENU 510
XmCR_ACTIVATE 510
XmCR_MAP 510
XmCR_UNMAP 510
XerrorCode
ICurrentThread 241
xHeight
IFontDialog 327
XmCR_ACTIVATE 510
XmCR_MAP 510
XmCR_UNMAP 510

Y

y
IPoint 658

Printed in U.S.A.

SC09-1971-00