

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

Version 3.5

Document Number S33H-5041-00

IBM VisualAge for C++ for Windows

S33H-5041-00

Open Class Library Reference
Volume III

Version 3.5

IBM

IBM VisualAge for C++ for Windows

S33H-5041-00

**Open Class Library Reference
Volume III**

Version 3.5

Note

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

First Edition (March 1996)

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

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

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

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

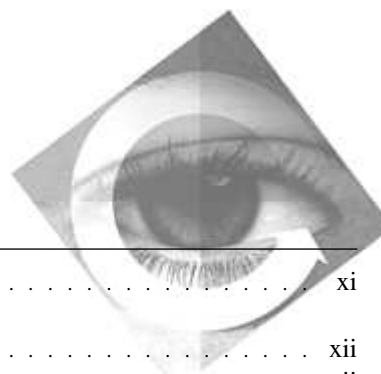
You can also send your comments by facsimile (attention: RCF Coordinator), or, you can send your comments electronically to IBM. See “Communicating Your Comments to IBM” for a description of the methods. This page immediately precedes the Readers’ Comment Form at the back of this publication.

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

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

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

Contents



Part 1. About this Book	xi
Notices	xii
Programming Interface Information	xii
Trademarks	xiii
About This Book	xiv
Who Should Use This Book	xiv
Conventions Used in This Book	xiv
How This Reference is Organized	xv
About the User Interface Class Library	xvi
User Interface Class Library Conventions	xix
File Names	xix
Class Names and Member Names	xxi
Function Return Types and Function Arguments	xxi
Using Obsolete or Ignored Member Functions	xxii
A Note about Samples and Examples	xxiv
Part 2. Description of Classes	1
Class Hierarchy by Category	2
IAnimatedButton	7
IAnimatedButton::Style	16
IBitmapControl	17
IBitmapControl::Style	25
ICanvas	26
ICanvas::Style	37
ICircularSlider	38
ICircularSlider::Style	50
ICircularSliderNotifyHandler	51

ICnrAllocator	54
ICnrBeginEditEvent	57
ICnrControlList	60
ICnrDate	62
ICnrDrawBackgroundEvent	63
ICnrDrawHandler	66
ICnrDrawItemEvent	72
ICnrEditEvent	76
ICnrEditHandler	81
ICnrEmphasisEvent	87
ICnrEndEditEvent	90
ICnrEnterEvent	93
ICnrEvent	96
ICnrHandler	98
ICnrHelpEvent	103
ICnrMenuHandler	106
ICnrObjectSet	111
ICnrQueryDeltaEvent	113
ICnrReallocStringEvent	116
ICnrScrollEvent	120
ICnrTime	123
ICollectionViewComboBox	124

ICollectionViewConstants	137
ICollectionViewListBox	140
IContainerColumn	153
IContainerControl	174
IContainerControl::Attribute	235
IContainerControl::ColumnCursor	236
IContainerControl::CompareFn	240
IContainerControl::FilterFn	242
IContainerControl::Iterator	244
IContainerControl::ObjectCursor	246
IContainerControl::Style	250
IContainerControl::TextCursor	251
IContainerControlNotifyHandler	255
IContainerObject	258
ICustomButton	270
ICustomButton::Style	280
ICustomButtonDrawEvent	281
ICustomButtonDrawHandler	285
IDDEAcknowledgeEvent	289
IDDEAcknowledgeExecuteEvent	293
IDDEAcknowledgePokeEvent	296
IDDEActiveServer	299

IDDEActiveServerSet	301
IDDEBeginEvent	303
IDDEClientAcknowledgeEvent	306
IDDEClientConversation	309
IDDEClientEndEvent	323
IDDEClientHotLinkEvent	326
IDDEClientHotLinkSet	329
IDDEDataEvent	331
IDDEEndEvent	334
IDDEEvent	337
IDDEExecuteEvent	340
IDDEPokeEvent	343
IDDERequestDataEvent	346
IDDEServerAcknowledgeEvent	349
IDDEServerHotLinkEvent	352
IDDESetAcknowledgeInfoEvent	355
IDDETopicServer	358
IDrawingCanvas	369
IDrawingCanvas::Style	376
IFileDialog	378
IFileDialog::Settings	390
IFileDialog::Style	396

IFileDialogEvent	398
IFileDialogHandler	400
IFlyOverHelpHandler	406
IFlyText	415
IFontDialog	419
IFontDialog::Settings	430
IFontDialog::Style	434
IFontDialogHandler	435
IGraphicPushButton	439
IGraphicPushButton::Style	450
IIconControl	451
IIconControl::Style	459
IInfoArea	460
IMultiCellCanvas	470
IMultiCellCanvas::Style	487
INotebook	489
INotebook::Cursor	525
INotebook::PageSettings	529
INotebook::PageSettings::Attribute	537
INotebook::Style	539
INotebookDrawItemEvent	540
INotebookNotifyHandler	543

IPageEvent	546
IPageHandle	549
IPageHandler	551
IPageHelpEvent	556
IPageRemoveEvent	559
IPageSelectEvent	562
ISetCanvas	565
ISetCanvas::Style	581
ISplitCanvas	583
ISplitCanvas::Style	595
IStringGenerator	597
IStringGeneratorFn	601
IStringGeneratorMemberFn	604
IStringGeneratorRefMemberFn	607
IToolBar	610
IToolBar::FrameCursor	627
IToolBar::Style	629
IToolBar::WindowCursor	630
IToolBarButton	632
IToolBarButton::Style	647
IToolBarContainer	648
IToolBarContainer::Style	654

IToolBarFrameWindow	655
IViewPort	660
IViewPort::Style	673
Glossary	675
Bibliography	692
Index	693

Part 1. About this Book

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



Notices

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

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

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

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

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

Programming Interface Information

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

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

Trademarks

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

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

Windows is a trademark of Microsoft Corporation.

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

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

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

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

Who Should Use This Book

About This Book

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

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

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

Who Should Use This Book

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

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

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

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

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

Conventions Used in This Book

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

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

Who Should Use This Book

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

How This Reference is Organized

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

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

Volume III contains the classes grouped in the following categories:

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

Volume IV contains the classes grouped in the following categories:

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

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

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

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



Portability Considerations



Motif Information



Presentation Manager Information



Windows Information



Win32s Information

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

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

About the User Interface Class Library

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

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

For example:

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

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

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

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

About the User Interface Class Library

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

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

About the User Interface Class Library

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

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

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

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

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

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

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

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

About the User Interface Class Library

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

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

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

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

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

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

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

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

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

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

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

User Interface Class Library Conventions

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

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

File Names

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

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

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

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

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

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

About the User Interface Class Library



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

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

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

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

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

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

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

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

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

About the User Interface Class Library

CPPWOU3I.DLL Multithreaded dynamic link library file containing the User Interface Base Library classes.

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

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

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

Class Names and Member Names

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

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

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

Function Return Types and Function Arguments

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

Function return types for the various functions are as follows:

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

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

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

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

About the User Interface Class Library

Instead, use the following:

```
if(isValid)
```

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

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

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

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

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

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

Function arguments are passed in the following ways:

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

Using Obsolete or Ignored Member Functions

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

Obsolete Functions

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

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

About the User Interface Class Library

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

```
#define IC_OBSOLETE_1    310

#define IC_OBSOLETE_2    400

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

Obsolete interface is then wrapped as follows:

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

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

There are several important guidelines regarding obsolete functions:

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

Ignored Functions

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

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

Samples and Examples

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

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

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

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

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

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

A Note about Samples and Examples

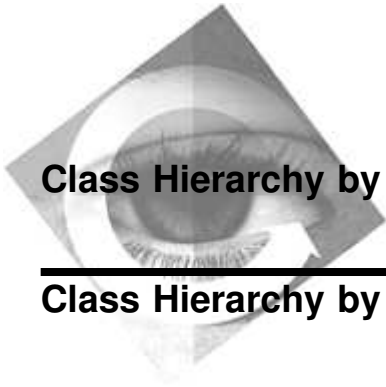
This *Open Class Library Reference* contains examples and references to samples. Samples, including the Hello World sample application, are shipped with the User Interface Class Library product in the following directory:

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

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

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

Part 2. Description of Classes



Class Hierarchy by Category

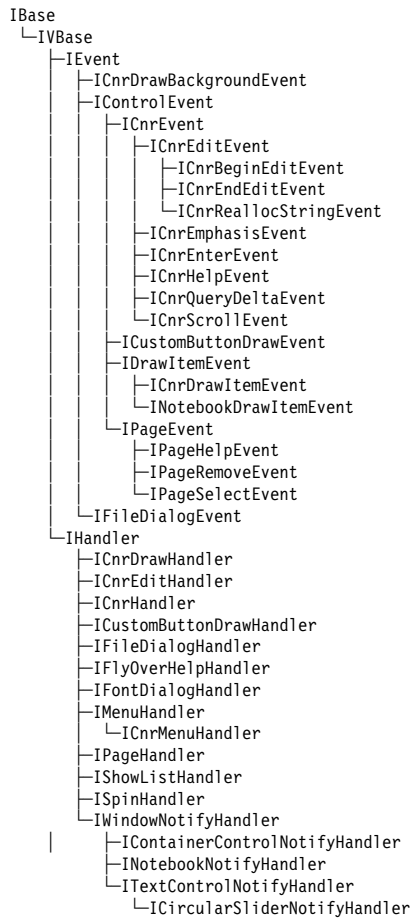
Class Hierarchy by Category

Advanced Control, Dialog, and Handler Classes

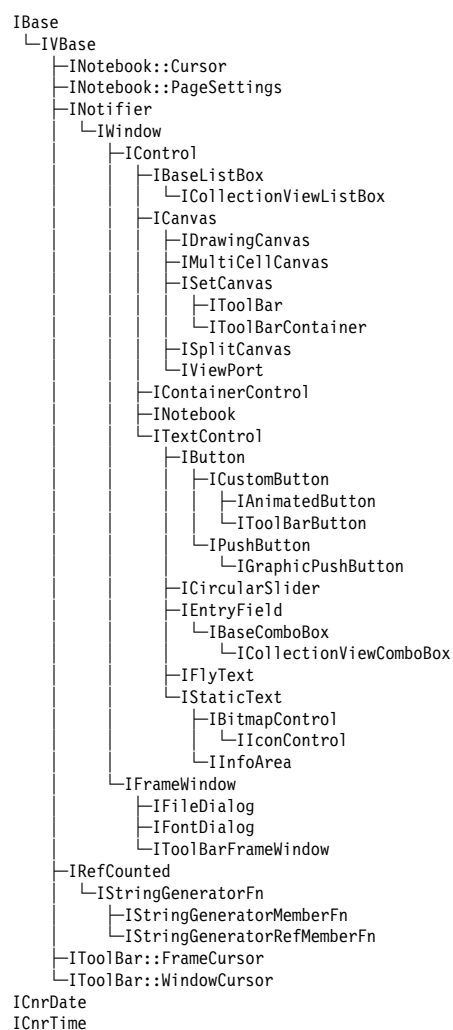
Provide support for the advanced controls, such as container, notebook, and toolbar, and for the font and file dialogs used by the applications you develop.

```
ISequence
└─ICnrObjectSet
ISet
└─ICnrControlList
IBase
├─IBitFlag
│   └─IContainerControl::Attribute
│       └─INotebook::PageSettings::Attribute
│           └─IAnimatedButton::Style
│               └─IBitmapControl::Style
│                   └─ICanvas::Style
│                       └─ICircularSlider::Style
│                           └─IContainerControl::Style
│                               └─ICustomButton::Style
│                                   └─IDrawingCanvas::Style
│                                       └─IFileDialog::Style
│                                           └─IFontDialog::Style
│                                               └─IGraphicPushButton::Style
│                                                   └─IIconControl::Style
│                                                       └─IMultiCellCanvas::Style
│                                                           └─INotebook::Style
│                                                               └─ISetCanvas::Style
│                                                                   └─ISplitCanvas::Style
│                                                                       └─IToolBar::Style
│                                                                           └─IToolBarButton::Style
│                                                                               └─IToolBarContainer::Style
│                                                                                   └─IViewport::Style
├─ICnrAllocator
├─IFileDialog::Settings
├─IFontDialog::Settings
├─IStringGenerator
├─IHandle
│   └─IPageHandle
├─IVBase
│   └─IContainerColumn
│       └─IContainerControl::ColumnCursor
│           └─IContainerControl::CompareFn
│               └─IContainerControl::FilterFn
│                   └─IContainerControl::Iterator
│                       └─IContainerControl::ObjectCursor
│                           └─IContainerControl::TextCursor
└─IContainerObject
```

Advanced Control, Dialog, and Handler Classes ...

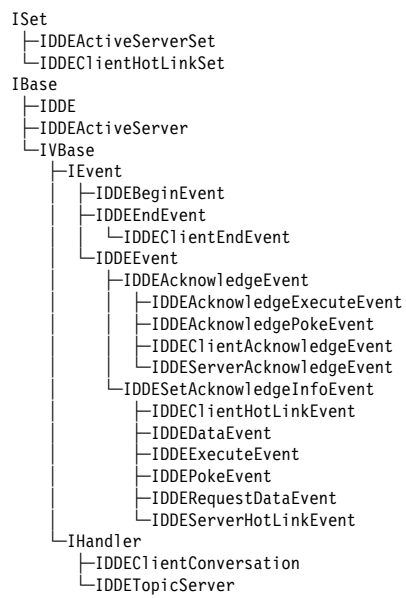


Advanced Control, Dialog, and Handler Classes ...



Dynamic Data Exchange Classes

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





IAAnimatedButton

Derivation	IBase
	IVBase
	INotifier
	IWindow
	IControl
	ITextControl
	IButton
	ICustomButton
	IAAnimatedButton

Inherited By None.

Header File ianimbut.hpp

Members	Member	Page	Member	Page
	Constructor	10	enableAnimateWhenLatched	8
	animateWhenLatched	14	isAnimatedWhenLatched	8
	animationRate	8	isAnimationStarted	8
	bitmap	9	latch	8
	bitmapCount	9	setAnimationRate	9
	calcMinimumSize	13	setBitmaps	10
	classDefaultStyle	14	setCurrentBitmapIndex	10
	convertToGUIStyle	11	setDefaultStyle	12
	currentBitmapIndex	10	startAnimation	9
	defaultStyle	11	stopAnimation	9
	disableAnimateWhenLatched	8	~IAAnimatedButton	11

The IAAnimatedButton class creates and manages the animated button control window. A set of bitmaps are specified for the button to use for animation. During animation, the button cycles through the bitmaps on a time interval.

The bitmap set can be specified by the application by providing the resource ID of the first bitmap and the number of bitmaps to be used. The bitmap set can also be specified by using one of the animated bitmap sets that are provided by the library for standard multimedia actions.

If the button text is not visible, the current animated bitmap is centered in the button, but it is not resized. If the button text is visible, the current animated bitmap is centered horizontally and positioned vertically along the top of the button with the text centered horizontally immediately below the bitmap. No clipping is done for the bitmap, so it can paint outside the button border if the button is too small.

IAnimatedButton

Public Functions

Animation

Use these members to control the animation of the button.

animationRate

Returns the current animation rate for the button. The animation rate is specified in thousandths of a second.

```
unsigned long  
animationRate() const;
```

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

disableAnimateWhenLatched

Removes the animateWhenLatched style for the button.

```
virtual IAnimatedButton&  
disableAnimateWhenLatched();
```

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

enableAnimateWhenLatched

Sets the animateWhenLatched style for the button.

```
virtual IAnimatedButton&  
enableAnimateWhenLatched( Boolean enable = true );
```

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

isAnimatedWhenLatched

Returns true if the animateWhenLatched style is set.

```
Boolean  
isAnimatedWhenLatched() const;
```

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

isAnimationStarted

Returns true if the button is currently animated.

```
Boolean  
isAnimationStarted() const;
```

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

latch

Puts the button in a latched or unlatched state based on the *latched* parameter. The value of the *refresh* parameter determines if the button is painted.

IAnimatedButton

```
virtual IAnimatedButton&
    latch( Boolean latched = true,
           Boolean refresh = true );
```

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

setAnimationRate

Sets the animation rate for the button in thousandths of a second. The default animation rate is one second.

The animation rate represents the amount of time that each bitmap is displayed when the button is animated.

Note: This class uses the ITimer class to control animation. Refer to the ITimer (Vol. II) class for more information.

```
virtual IAnimatedButton&
    setAnimationRate( unsigned long newRate = 1000 );
```

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

startAnimation

Starts the animation of the button. If you specify an index, the bitmap at the index is the first bitmap displayed.

```
virtual IAnimatedButton&
    startAnimation( unsigned long index = 0 );
```

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

Exceptions	
InvalidParameter	The specified bitmap index is invalid.

stopAnimation

Stops the animation of the button.

```
virtual IAnimatedButton&
    stopAnimation();
```

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

Bitmaps

Use these members to set and query the bitmaps that are displayed for a button.

bitmap Returns the handle of the specified bitmap.

```
virtual IBitmapHandle
    bitmap( unsigned long index = 0 ) const;
```

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

bitmapCount Returns the number of bitmaps specified for the button.

IAnimatedButton

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

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

currentBitmapIndex

Returns the index of the bitmap that is currently displayed for the button.

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

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

setBitmaps Sets the bitmaps to be used for the button. If this function is called more than once, the button uses only the bitmaps specified in the last call.

1

```
virtual IAnimatedButton&  
    setBitmaps( AnimatedBitmaps bitmaps );
```

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

This function loads the specified resource bitmap set that the library provides.

Exceptions	
InvalidParameter	The animated bitmap set identifier is invalid.

2

```
virtual IAnimatedButton&  
    setBitmaps( const IResourceId& firstBitmap,  
                unsigned long count );
```

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

The bitmaps must be sequential, starting with the value specified as the first bitmap ID and continuing for the specified bitmap count.

setCurrentBitmapIndex

Sets the index of the bitmap that is currently displayed for the button.

```
virtual IAnimatedButton&  
    setCurrentBitmapIndex( unsigned long index = 0 );
```

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

Constructors

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

IAnimatedButton

IAnimatedButton

1 IAnimatedButton(unsigned long id, Win PM Motif
IWindow* parent, Y Y N
IWindow* owner,
const IRectangle& initial = IRectangle (),
const Style& style = defaultStyle ());

Creates an IAnimatedButton with the specified window ID, parent and owner windows, screen position and size, and window style.

Exceptions	
InvalidParameter	The parent window pointer is invalid.

2 IAnimatedButton(const IWindowHandle& handle); Win PM Motif
Y Y N

Creates an IAnimatedButton object for the specified button control that has a user-button style.

~IAnimatedButton

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

Styles

Use these style members to set and query IAnimatedButton styles. You can use these styles with the styles defined by the following nested classes:

IWindow::Styles (Vol. II)
IControl::Styles (Vol. II)
IButton::Styles (Vol. II)
ICustomButton::Styles (p. 277)

convertToGUIStyle

Converts style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles defined by the User Interface Class Library can be returned by setting the *extendedOnly* parameter to true.

virtual unsigned long Win PM Motif
convertToGUIStyle(const IBitFlag& style, Y Y N
Boolean extendedOnly = false) const;

defaultStyle Returns the default style. The default style is classDefaultStyle unless you have changed the style using setDefaultStyle.

IAnimatedButton

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

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

setDefaultStyle

Sets the default style for all subsequent animated buttons.

style Use the styles provided by IAnimatedButton::Style to specify the default style.

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

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

Inherited Public Functions

ICustomButton		
backgroundColor	isAutoLatchEnabled	resetLatchedBackgroundColor
convertToGUIStyle	isLatched	resetLatchedForegroundColor
defaultStyle	isLatchedBackgroundColorHalfTone	setDefaultStyle
disableAutoLatch	isLatchingEnabled	setLatchedBackgroundColor
disableLatching	latch	setLatchedForegroundColor
enableAutoLatch	latchedBackgroundColor	setUserData
enableLatching	latchedForegroundColor	unlatch

IButton		
allowsMouseClickedFocus	disableMouseClickedFocus	highlight
backgroundColor	enableMouseClickedFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup

IAnimatedButton

IControl		
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

These members manage the layout of an animated button.

calcMinimumSize

Returns the recommended minimum size of this animated button control. The size is based on the bitmap, button text, and the current font.

virtual ISize	Win	PM	Motif
calcMinimumSize() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

ICustomButton		
calcMinimumSize		

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

IAnimatedButton

Public Data

Styles

Use these style members to set and query IAnimatedButton styles. You can use these styles with the styles defined by the following nested classes:

- IWindow::Styles (Vol. II)
- IControl::Styles (Vol. II)
- IButton::Styles (Vol. II)
- ICustomButton::Styles (p. 277)

animateWhenLatched

Starts animation automatically when the button is in the latched state and stops animation when the button is in the unlatched (default) state.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
animateWhenLatched;	<i>Y</i>	<i>Y</i>	<i>N</i>

classDefaultStyle

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

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

Inherited Public Data

ICustomButton		
autoLatch	classDefaultStyle	latchable

IButton		
buttonClickId	noPointerFocus	

ITextControl		
textId		

IControl		
group	tabStop	

IAnimatedButton

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IAnimatedButton contains the following nested classes:

IAnimatedButton::Style (see page 16)

AnimatedBitmaps

```
AnimatedBitmaps {
    rewind,      stop,      pause,      play,      fastForward,
    record,      mute,      trackAdvance, trackReverse, stepForward,
    stepBackward, scanForward, scanBackward, eject,      volumeUp,
    volumeDown
};
```

Enumeration that defines identifiers for each of the provided animated bitmap resource sets. IBM User Interface Class Library provides these bitmaps for standard multimedia operations.

IAnimatedButton::Style

IAnimatedButton::Style

Derivation

IBase
IBitFlag
IAnimatedButton::Style

Inherited By None.

Header File ianimbut.hpp

The IAnimatedButton::Style class provides a set of valid styles for the static member functions of the class IAnimatedButton.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IBitmapControl

Derivation

```

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IStaticText
              IBitmapControl
  
```

Inherited By IIconControl

Header File ibmpctl.hpp

Members	Member	Page	Member	Page
	Constructor	18	moveSizeTo	21
	bitmap	17	setBitmap	18
	calcMinimumSize	22	setDefaultStyle	21
	classDefaultStyle	23	setLayoutDistorted	20
	convertToGUIStyle	20	sizeToBitmap	23
	defaultStyle	21	~IBitmapControl	20

The IBitmapControl class creates and manages bitmap controls. A bitmap control draws a bitmap for its background.



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

Public Functions

Bitmaps

Use these members to query and set the bitmap for the bitmap control. You can query the handle of the current bitmap. You can change the bitmap of the bitmap control using these functions.

bitmap Returns the handle to the bitmap.

IBitmapHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bitmap() const;	Y	Y	Y

IBitmapControl

setBitmap Changes the bitmap used by the IBitmapControl object.

1	<code>virtual IBitmapControl& setBitmap(unsigned long bitmapId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Set the bitmap using the default resource library with the specified bitmap ID.

2	<code>virtual IBitmapControl& setBitmap(const IResourceId& bitmapId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

Set the bitmap by specifying an IResourceId. If you want to load the bitmap from a specific resource library, use this function.

3	<code>virtual IBitmapControl& setBitmap(const IBitmapHandle& handle);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

Set the bitmap using an existing bitmap handle.

4	<code>virtual IBitmapControl& setBitmap(ISystemBitmapHandle::Identifier bitmap);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Set the bitmap using a system bitmap.

Constructors

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

IBitmapControl

1	<code>IBitmapControl(unsigned long id, IWindow* parent, IWindow* owner, const IResourceId& bitmapId, const IRectangle& initial = IRectangle (), const Style& style = defaultStyle ());</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Create a bitmap control using the specified window ID, parent and owner windows, resource library with its specified bitmap ID, screen position and size, and window style. If you have not already loaded it from the .DLL or .EXE of your choice, use this constructor.

IBitmapControl

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

2 IBitmapControl(unsigned long id, Win PM Motif
IWindow* parent, Y Y Y
IWindow* owner,
unsigned long bitmapId,
const IRectangle& initial = IRectangle (),
const Style& style = defaultStyle ());

Create a bitmap control with the specified ID, parent and owner windows, default resource library with its specified bitmap ID, and window style. If you have not already loaded the bitmap and you want load it from the default resource library, use this constructor.

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

3 IBitmapControl(Win PM Motif
unsigned long id, Y Y Y
IWindow* parent,
IWindow* owner,
const IBitmapHandle& bitmapId = IBitmapHandle (),
const IRectangle& initial = IRectangle (),
const Style& style = defaultStyle ());

Create a bitmap control with the specified window ID, parent and owner windows, bitmap, screen position and size, and window style. If you already have a bitmap handle, use this constructor.

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

4 IBitmapControl(unsigned long id, Win PM Motif
IWindow* parent, Y Y N
IWindow* owner,
ISystemBitmapHandle::Identifier bitmapId,
const IRectangle& initial = IRectangle (),
const Style& style = defaultStyle ());

Create a bitmap control with the specified window ID, parent and owner windows, specified system bitmap, screen position and size, and window style. If you want to use a system bitmap, use this constructor.

IBitmapControl

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

5 IBitmapControl(unsigned long id, Win PM Motif
IWindow* parentDialog); Y Y Y

Create an IBitmapControl object for the specified bitmap dialog control.

6 IBitmapControl(const IWindowHandle& handle); Win PM Motif
Y Y Y

Create an IBitmapControl object for the specified bitmap control's window handle.

~IBitmapControl

virtual Win PM Motif
~IBitmapControl(); Y Y Y

Layout Support

Layout is information used by the canvas classes to provide dialog-like behavior.

setLayoutDistorted

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

virtual IBitmapControl& Win PM Motif
setLayoutDistorted(unsigned long layoutAttributesOn, Y Y N
unsigned long layoutAttributesOff);

Styles

Use these style members to provide valid styles for IBitmapControl::setDefaultStyle and for the constructor of the IBitmapControl (p. 17) class.

convertToGUIStyle

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

IBitmapControl

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

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

defaultStyle Returns the default style. This style is classDefaultStyle (p. 23) unless you have changed it using setDefaultStyle (p. 21).

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

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

setDefaultStyle

Sets the default style for all subsequent bitmap controls.

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

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

Window Positioning

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

moveSizeTo Changes the position and size of the window.

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

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



This IWindow function is overridden to handle bitmap sizing and positioning in Motif.

Inherited Public Functions

IStaticText		
alignment	enableFillBackground	isStrikeout
backgroundColor	enableHalftone	isUnderscore
convertToGUIStyle	enableStrikeout	limit
defaultStyle	enableUnderscore	moveSizeTo
disableFillBackground	fillColor	resetFillColor
disableHalftone	foregroundColor	setAlignment
disableStrikeout	hasFillBackground	setDefaultStyle
disableUnderscore	isHalftone	setFillColor

IBitmapControl

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout is information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size this bitmap control can be, based on the actual size of the currently set bitmap.

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

Win	PM	Motif
Y	Y	Y

Exceptions	
IAccessError	The call to determine the size of the bitmap failed. The bitmap may be corrupted.

Inherited Protected Functions

IStaticText		
calcLimitSize	calcMinimumSize	passEventToOwner

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these style members to provide valid styles for IBitmapControl::setDefaultStyle and for the constructor of the IBitmapControl (p. 17) class.

classDefaultStyle

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

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

sizeToBitmap

Sizes the window to the size of the currently set bitmap.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sizeToBitmap;	Y	Y	Y

Inherited Public Data

IStaticText		
border3D	fillColorId	right
bottom	halftone	strikeout
center	halftoneId	strikeoutId
classDefaultStyle	left	top
fillBackground	limitId	underscore

IBitmapControl

IStaticText		
fillBackgroundId	mnemonic	underscoreId

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IBitmapControl contains the following nested classes:

IBitmapControl::Style (see page 25)

IBitmapControl::Style



IBitmapControl::Style

Derivation IBase
 IBitFlag
 IBitmapControl::Style

Inherited By None.

Header File ibmpctl.hpp

The nested class IBitmapControl::Style provides a set of valid styles for the IBitmapControl (p. 17) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

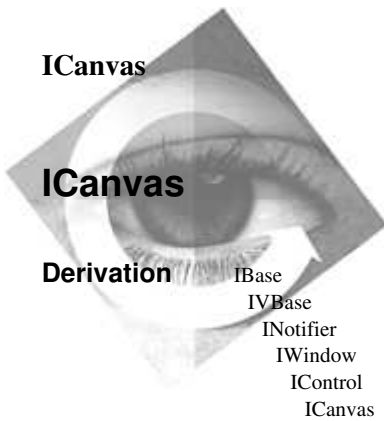
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By

IDrawingCanvas	ISplitCanvas
IMultiCellCanvas	IView
ISetCanvas	IViewPort

Header File icanvas.hpp

Members				
	Member	Page	Member	Page
	Constructor	27	layoutSize	34
	areChildrenReversed	33	matchForMnemonic	29
	backgroundColor	27	origDefaultButtonHandle	29
	calcMinimumSize	34	passEventToOwner	32
	classDefaultStyle	35	registerCallbacks	33
	convertToGUIStyle	30	setDefaultStyle	30
	defaultPushButton	28	setFont	29
	defaultStyle	30	setLayoutDistorted	29
	fixupChildren	34	setLayoutSize	34
	initialize	31	unregisterCallbacks	33
	isTabStop	29	~ICanvas	28
	layout	34		

The ICanvas class provides dialog-like behavior for its child windows. This support includes the following:

- Transferring the input focus to a child window that can accept it
- Handling tab and cursor movement keys by moving the input focus between child windows
- Allowing the client window or the frame window to be the central point for processing command events
- Allowing the frame window to change the appearance of the mouse pointer when the pointer is over the canvas or a child window of the canvas

Additionally, this class provides the above behavior for all derived classes.

ICanvas

You can use a canvas as the client window for a frame window, as a frame extension, or as the child window of another canvas. To specify the client window for an `IFrameWindow`, use `IFrameWindow::setClient` (Vol. II). To add a canvas as a frame extension, use `IFrameWindow::addExtension` (Vol. II).

Unlike other canvas classes, `ICanvas` does not alter the size or position of its child windows. Others, such as `IMultiCellCanvas` (p. 470), `ISetCanvas` (p. 565), `ISplitCanvas` (p. 583), and `IViewPort` (p. 660), provide special layout rules for sizing and positioning their child windows.

Public Functions

Colors

Use these members to query, set, and reset the color of a specified region of the window.

backgroundColor

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

<code>virtual IColor</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>backgroundColor() const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>



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

Constructors

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

ICanvas

<code>ICanvas(unsigned long windowIdentifier,</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code> IWindow* parent,</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<code> IWindow* owner,</code>			
<code> const IRectangle& initial = IRectangle (),</code>			
<code> const Style& style = defaultStyle ());</code>			

Use this constructor to create a canvas window.

windowIdentifier

The window identifier of the canvas you are constructing.

We recommend that you do the following:

ICanvas

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

Presentation Manager Notes: Do not use FID_XXX values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

parent The parent window of the canvas you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.

owner The owner window of the canvas you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.

Motif Notes: The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial The initial position and size of the canvas you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.

style The window's characteristics. This value can be a combination of ICanvas::Style (p. 35) and IWindow::Style (Vol. II) objects. Optional.

~ICanvas

```
virtual  
~ICanvas();
```

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

Dialog Behavior Support

Use these members to provide dialog-like keyboard support for the canvas classes.

defaultPushButton

Returns the current default push button if this canvas is in its parent window chain.

Pressing the Enter key causes the default push button to be selected.

ICanvas

<code>virtual IWindowHandle defaultPushButton() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

isTabStop 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.

<code>virtual Boolean isTabStop() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

matchForMnemonic

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

<code>virtual IWindowHandle matchForMnemonic(unsigned short character) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

origDefaultButtonHandle

Returns the push button originally identified as the default. This push button is made the default when the window with the input focus is not a push button.

Pressing the Enter key causes the default push button to be selected.

<code>IWindowHandle origDefaultButtonHandle() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Font Functions

Use these members to query and set the font of a specified region of the window.

setFont Sets the canvas's font to the indicated font.

<code>virtual IWindow& setFont(const IFont& font);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>N</i>	<i>N</i>

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

setLayoutDistorted

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.

ICanvas

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` (Vol. II).

`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.

<code>virtual ICanvas&</code>			
<code> setLayoutDistorted(unsigned long layoutAttributesOn,</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code> unsigned long layoutAttributesOff);</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

Style

Use these members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the nested class `IWindow::Style` (Vol. II).

Once you have constructed an `ICanvas` object, you can use `ICanvas` and `IWindow` member functions to query and change individual styles.

convertToGUIStyle

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

<code>virtual unsigned long</code>			
<code> convertToGUIStyle(const IBitFlag& style,</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code> Boolean extendedOnly = false) const;</code>	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultStyle Returns the default style. The default style is `classDefaultStyle` (p. 35) unless you have changed the style using `setDefaultStyle` (p. 30).

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

setDefaultStyle

Sets the default style for all subsequent canvases.

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

ICanvas

style A combination of ICanvas::Style (p. 35) and IWindow::Style (Vol. II) objects.

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Constructor Implementation

ICanvas provides functions to help you implement the constructors of derived classes.

initialize Creates a presentation system window and calls IWindow::startHandlingEventsFor (Vol. II). Call this function from a constructor of a class derived from ICanvas if the constructor calls the protected ICanvas constructor.

```
ICanvas&
  initialize( unsigned long windowIdentifier,
              IWindow* parent,
              IWindow* owner,
              const IRectangle& initialRect,
              unsigned long style,
              unsigned long extendedStyle );
```

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

ICanvas

windowIdentifier

The identifier of the canvas window you are creating.

parent

The parent window of the canvas window you are creating. You must specify a parent window. The parent window is primarily used for visible relationships.

owner

The owner window of the canvas window you are creating. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.

initialRect

The initial position and size of the window you are creating. See ICoordinateSystem (Vol. II) for additional information.

style

The window's characteristics. This value is a combination of flags recognized by the presentation system. You can generate this value by passing a Style object to the convertToGUIStyle (p. 30) function of the canvas object.

extendedStyle

Additional characteristics of the window. This value is a combination of flags that you can also generate using convertToGUIStyle.

PM

The *extendedStyle* parameter is ignored.

Exceptions	
InvalidParameter	You must specify a nonzero value for <i>parent</i> .

Constructors

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

ICanvas

ICanvas();

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

Derived classes that create their own windows should call this ICanvas constructor.

Event-Handling Implementation

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

passEventToOwner

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

ICanvas

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

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

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 you override this function in a class derived from ICanvas, your implementation of registerCallbacks must call ICanvas::registerCallbacks to register the applicable callbacks and X event handlers.

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

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

unregisterCallbacks

Removes callbacks and X events added by registerCallbacks.

If you override this function in a class derived from ICanvas, your implementation of unregisterCallbacks must call ICanvas::unregisterCallbacks for clean-up to occur.

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

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

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

areChildrenReversed

States whether the order in which child windows are returned by the class IWindow::ChildCursor (Vol. II) 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.

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

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

ICanvas

calcMinimumSize

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

<code>virtual ISize calcMinimumSize() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

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.

<code>virtual IWindowPosBuffer fixupChildren();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

layout

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

<code>virtual ICanvas& layout();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

layoutSize

Returns the size needed to show all child windows.

<code>ISize layoutSize() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

setLayoutSize

Sets the size needed to show all child windows.

<code>ICanvas& setLayoutSize(const ISize& size);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Style

Use these members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the nested class IWindow::Style (Vol. II).

Once you have constructed an ICanvas object, you can use ICanvas and IWindow member functions to query and change individual styles.

classDefaultStyle

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

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

Inherited Public Data

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ICanvas

Nested Classes

ICanvas contains the following nested classes:

ICanvas::Style (see page 37)



ICanvas::Style

Derivation IBase
 IBitFlag
 ICanvas::Style

Inherited By None.

Header File icanvas.hpp

The nested class ICanvas::Style provides a set of valid canvas styles for the ICanvas::defaultStyle (p. 30) and ICanvas::setDefaultStyle (p. 30) functions, and for the constructor of the class ICanvas (p. 26). You can use these styles with the styles defined by the nested class IWindow::Style (Vol. II).

Once you have constructed an ICanvas object, you can use ICanvas and IWindow member functions to query and change individual styles.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icslider.hpp

Members	Member	Page	Member	Page
	Constructor	42	proportionalTicks	48
	armRange	39	radius	40
	buttons	46	rotationIncrement	40
	calcMinimumSize	45	setArmRange	40
	circularArm	46	setDecrementBitmaps	40
	classDefaultStyle	47	setDefaultStyle	44
	convertToGUIStyle	44	setIncrementBitmaps	41
	defaultStyle	44	setRotationIncrement	41
	displayValue	47	setTickSpacing	41
	enableNotification	44	setValue	41
	full360	47	tickSpacing	41
	jumpToPointer	47	trackId	46
	label	47	value	42
	midpoint	47	valueId	46
	noTicks	47	~ICircularSlider	43

The ICircularSlider class provides a control to represent a circular slider, which allows a user to set, display, or modify a value by rotating the slider arm. The circular slider emulates the actual controls of stereo and video components.

You can use a circular slider to allow users to control the balance, bass, volume, and treble. If they click on one of the tick marks, then the circular slider rotates to the new position. Or they can grab the slider arm with the mouse and rotate it. The circular slider also has plus and minus buttons for moving the slider arm.

Use the ISliderArmHandler (Vol. II) class to process the events that are generated as the user is rotating the circular slider or when the user stops rotating the circular slider. This position change event is also generated when the user just clicks on one of the tick marks.

ICircularSlider

And finally, use the IFocusHandler (Vol. II) class to process any focus changes. The following figure shows an ICircularSlider with the default, midpoint, and circular arm styles.

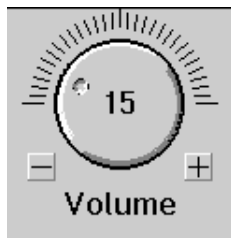


Figure 1. An ICircularSlider with Default, Midpoint, and Circular Arm Styles

The following figure shows an ICircularSlider with the default and proportional ticks styles.

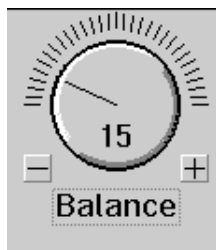


Figure 2. An ICircularSlider with the Default and Proportional Ticks Styles



You must install the Multimedia Presentation Manager to use ICircularSlider under the OS/2 2.1 operating system.

Public Functions

Arm Operations

Use these functions to set and query attributesslider arm. They can query and set the arm range, the rotational increment, and tick spacing. Also, you can change the increment and decrement bitmaps.

armRange Returns the range of values over which the arm can be moved. This range contains the minimum and maximum value of the circular slider.

```
IRange  
armRange() const;
```

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

ICircularSlider

radius Returns the radius of the dial in pixels. This value is indirectly set by sizing the control.

<code>unsigned long radius() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

rotationIncrement

Returns the increment that the arm is rotated. This is the number by which the current value is incremented or decremented when the user selects one of the circular slider increment or decrement buttons.

<code>unsigned long rotationIncrement() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setArmRange

Sets the range of values over which the arm can be rotated. This range contains the minimum and maximum value of the circular slider. The values returned to the circular slider are based on the position of the arm inside of this range. If you set a range of 0 to 100 and the arm is in the middle, then calling `value` (p. 42) returns 50 as the current value. The circular slider supports a minimum value of -32K and a maximum value of 32K.

<code>ICircularSlider& setArmRange(const IRange& range);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDecrementBitmaps

Sets the bitmaps used for decrement buttons of the circular slider. They are on the left-hand side of the slider.

1 <code>ICircularSlider& setDecrementBitmaps(const IResourceId& leftUp, const IResourceId& leftDown);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function to load the bitmaps from the resource file.

2 <code>ICircularSlider& setDecrementBitmaps(const IBitmapHandle& leftUp, const IBitmapHandle& leftDown);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Use this function if you have bitmap handles.

ICircularSlider

setIncrementBitmaps

Sets the bitmaps used for increment buttons of the circular slider. They are on the right-hand side of the slider.

1	ICircularSlider& setIncrementBitmaps(const IBitmapHandle& rightUp, const IBitmapHandle& rightDown);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Use this function if you have bitmap handles.

2	ICircularSlider& setIncrementBitmaps(const IResourceId& rightUp, const IResourceId& rightDown);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	---	-------------------------------	------------------------------	---------------------------------

Use this function to load the bitmaps from the resource file.

setRotationIncrement

Sets the scroll increment. This is the number by which the current value of the circular slider is incremented or decremented when the user selects one of the circular slider increment or decrement buttons.

ICircularSlider& setRotationIncrement(unsigned long increment);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

setTickSpacing

Sets the tick mark increment of the control. This represents the value between the tick marks around the circular slider.

ICircularSlider& setTickSpacing(unsigned long tick);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

setValue

Sets the value of the circular slider. This value must be within the current range. The position of the arm is rotated to this value.

ICircularSlider& setValue(long value);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

tickSpacing

Returns the increment used to draw the tick marks. This represents the value between the tick marks around the circular slider.

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

ICircularSlider

value Returns the current value of the circular slider. This value is based on where the arm is within the current range.

long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
value() const;		Y	Y	N

Constructors

You can construct and destruct objects of this class.

ICircularSlider

1	ICircularSlider(const IWindowHandle& handle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

You can construct an object of this class by specifying a window handle.

handle The window handle of the circular slider control.

2	ICircularSlider(unsigned long Identifier, IWindow* parent, IWindow* owner, const IRectangle& initial = IRectangle (), const Style& style = defaultStyle ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

You can construct objects of this class from the following:

identifier Window identifier of the circular slider you construct.

We recommend that you do one of the following:

- Assign unique identifiers to all windows in the same frame window. “In the same window” means that the frame window is the first frame in its parent window chain.
- Assign the client window a window identifier of IC_FRAME_CLIENT_ID.

parent The parent window of the circular slider you construct. You must specify a parent window. This constructor throws an InvalidParameter exception if you pass an IWindow* of 0. The parent window is primarily used for visible relationships.

owner The owner window of the circular slider you construct.

ICircularSlider

- initial* The initial position and size of the circular slider you construct. The initial position is the lower-left corner of the circular slider relative to the lower-left corner of the parent window. Optional.
- style* Use the styles provided by ICircularSlider Style (p. 46) to specify the control's styles. Optional.

PM The window identifier is limited to the range 0 through 65535. The owner window is primarily used for routing notification events and unprocessed messages. The OS/2 operating system also uses the owner window chain to inherit colors.

3 ICircularSlider(unsigned long Identifier, Win PM Motif
IWindow* parentAndOwner); Y Y N

You can construct objects of this class from the following:

identifier Window identifier of the circular slider you construct.

We recommend that you do one of the following:

- Assign unique identifiers to all windows in the same frame window. "In the same window" means that the frame window is the first frame in its parent window chain.
- Assign the client window a window identifier of IC_FRAME_CLIENT_ID.

parentAndOwner

The parent and owner window of the circular slider you construct. You must specify a parent window. This constructor throws an InvalidParameter exception if you pass an IWindow* of 0. The parent window is primarily used for visible relationships.

PM The window identifier is limited to the range 0 through 65535. The owner window is primarily used for routing notification events and unprocessed messages. The OS/2 operating system also uses the owner window chain to inherit colors.

~ICircularSlider

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

Observer Notification

Observer notification members implement the public INotifier protocol for the ICircularSlider class.

ICircularSlider

enableNotification

Enables the circular slider control to send notifications to any observer objects.

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

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

Styles

Use these style members to set and query circular slider styles, the default style for this class, and to convert the style into a form that is recognizable to the underlying GUI. You can use these styles with the styles defined by the following nested classes:

- IWindow Style (Vol. II)
- IControl Style (Vol. II)

convertToGUIStyle

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

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

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

defaultStyle Returns the current default style. This is the same as classDefaultStyle unless setDefaultStyle has been called.

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

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

setDefaultStyle

Sets the default style for all subsequent circular sliders.

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

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

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text

ICircularSlider

ITextControl		
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Minimum Size Calculation

These members are called when the circular slider is put into a canvas. The canvas uses the minimum size to calculate how small the circular slider can be when it is displayed.

calcMinimumSize

Returns an ISize that indicates the minimum size that the circular slider can be. This is used by the ICanvas hierarchy of classes. The minimum size also uses the current font and circular slider text in its calculations.

virtual ISize	Win	PM	Motif
calcMinimumSize() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

ICircularSlider

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

This INotificationId string is used for all notifications that ICircularSlider provides to its observers.

trackId Notification identifier provided to observers when the circular slider is manipulated with the mouse. ICircularSlider provides the current value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const
trackId;

WinPMMotif
Y Y N

valueId Notification identifier provided to observers when the value of the circular slider control changes. ICircularSlider provides the new value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const
valueId;

WinPMMotif
Y Y N

Styles

Use these style members to set and query circular slider styles, the default style for this class, and to convert the style into a form that is recognizable to the underlying GUI. You can use these styles with the styles defined by the following nested classes:

IWindow Style (Vol. II)
IControl Style (Vol. II)

buttons Displays increment and decrement buttons.

static const Style
buttons;

WinPMMotif
Y Y N

circularArm Draws a small circle for the arm, rather than a line.

static const Style
circularArm;

WinPMMotif
Y Y N

ICircularSlider

classDefaultStyle

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

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

displayValue Displays the value on the dial.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
displayValue;	<i>Y</i>	<i>Y</i>	<i>N</i>

full360 Permits the rotational range to extend 360 degrees. This forces the displayValue style to be off, which keeps the slider arm from corrupting the number value.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
full360;	<i>Y</i>	<i>Y</i>	<i>N</i>

jumpToPointer

Causes the circular slider to change immediately (jump) to the value indicated by the position of the mouse. If this style is not present, then the circular slider rotates by the rotational increment to the value indicated by the position of the mouse

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
jumpToPointer;	<i>Y</i>	<i>Y</i>	<i>N</i>

label Displays title text below the dial.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
label;	<i>Y</i>	<i>Y</i>	<i>N</i>

midpoint Makes the midpoint tick mark larger. This tick mark is not visible if the noTicks style is set.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
midpoint;	<i>Y</i>	<i>Y</i>	<i>N</i>

noTicks Turns off the tick marks.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noTicks;	<i>Y</i>	<i>Y</i>	<i>N</i>

ICircularSlider

proportionalTicks

Allows the length of the tick marks to be calculated as a percentage of the radius; otherwise, they are set to a fixed length.

```
static const Style
    proportionalTicks;
```

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

Inherited Public Data

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ICircularSlider contains the following nested classes:

ICircularSlider::Style (see page 50)



ICircularSlider::Style

ICircularSlider::Style

Derivation IBase
IBitFlag
ICircularSlider::Style

Inherited By None.

Header File icslider.hpp

The nested class **ICircularSlider::Style** provides a set of valid circular slider styles for the **defaultStyle** and **setDefaultStyle** functions, and for the constructor of the class **ICircularSlider** (p. 38).

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ICircularSliderNotifyHandler



ICircularSliderNotifyHandler

Derivation

- IBase
- IVBase
- IHandler
- IWindowNotifyHandler
- ITextControlNotifyHandler
- ICircularSliderNotifyHandler

Inherited By None.

Header File icsliden.hpp

Members	Member	Page
	Constructor	51
	dispatchHandlerEvent	52
	~ICircularSliderNotifyHandler	52

The ICircularSliderNotifyHandler class processes events for all classes of circular sliders.

This class handles events that require the circular slider class to generate a notification. If notifications are enabled for this class, a notification is generated and sent to all observers when the proper conditions for the specific notification exist.

Public Functions

Constructors

Use these functions to construct and destruct objects of the ICircularSliderNotifyHandler class.

ICircularSliderNotifyHandler

Provides the default constructor.

ICircularSliderNotifyHandler();

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

ICircularSliderNotifyHandler

~ICircularSliderNotifyHandler

```
virtual  
~ICircularSliderNotifyHandler();
```

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

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Processing

Event-processing members evaluate an event to determine if it is appropriate for this handler object to process.

dispatchHandlerEvent

Notifies the circular slider observers if the following event is received:

circular slider change event

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

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

Inherited Protected Functions

ICircularSliderNotifyHandler

ITextControlNotifyHandler		
dispatchHandlerEvent		

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrAllocator

ICnrAllocator

Derivation IBase
ICnrAllocator

Inherited By None.

Header File icnrobj.hpp

Members				
	Member	Page	Member	Page
	Constructor	54	nextAvailable	55
	first	55	updateForInsert	56
	initialized	55	~ICnrAllocator	55
	next	55		

The ICnrAllocator class allocates and manages multiple container items that can be added to an IContainerControl. ICnrAllocator allows you to create an unlimited number of objects at once and then add any number of them to a container with one call. This method of allocating and inserting multiple container objects greatly reduces the time it takes to load a large number of items into a container.

As items from the ICnrAllocator are added to an IContainerControl via IContainerControl::addObjects or IContainerControl::addObjectsAfter, the ICnrAllocator keeps track of all used and unused items it is managing.

The ICnrAllocator destructor frees the memory occupied by any unused container items allocated by the constructor but not added to any IContainerControl.



A native Windows container (that is, a container constructed without the pmCompatible style) does not allow multiple insertion of container objects with one call. The ICnrAllocator is supported but does add each object separately in this case. The pmCompatible container does insert all the objects at once.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrAllocator Construct this object by providing the number of container items you want to allocate and the size of the object you will be adding to an IContainerControl.

ICnrAllocator

```
ICnrAllocator( unsigned long num,  
               unsigned long size );
```

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

~ICnrAllocator

```
virtual  
~ICnrAllocator();
```

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

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Object Information

These members give access to the objects allocated by ICnrAllocator.

first Returns the first item of the list.

```
void*  
first() const;
```

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

initialized Returns the number of items initialized.

```
unsigned long  
initialized() const;
```

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

next Returns the next item of the item passed in.

```
void*  
next( void* pRec );
```

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

nextAvailable Returns the next available item that has not been initialized.

```
void*  
nextAvailable();
```

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

ICnrAllocator

updateForInsert

Updates the state of an ICnrAllocator object after addObject or addObjectAfter is called.

ICnrAllocator&
updateForInsert();

Win
Y

PM
Y

Motif
I

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrBeginEditEvent

Derivation

- IBase
- IVBase
- IEvent
- IControlEvent
- ICnrEvent
- ICnrEditEvent
- ICnrBeginEditEvent

Inherited By None.

Header File icnreevt.hpp

Members	Member	Page
	Constructor	57
	currentText	58
	~ICnrBeginEditEvent	58

The ICnrBeginEditEvent class retrieves information about a container direct edit event. Objects of this class are dispatched when editing begins in one of the editable fields of the container. The event provides information about the text being edited as well as the type of text (heading, icon, or details view) being edited. See ICnrEditHandler::beginEdit (p. 83) for information about that function.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrBeginEditEvent

Constructs an object from an IControlEvent object. Construct ICnrBeginEditEvent by calling ICnrEditHandler::dispatchHandlerEvent from an IControlEvent in response to a request to begin editing. Such an occurrence is either as a result of user interaction, or it is under program control. See IControlEvent (Vol. II) and ICnrEditHandler::dispatchHandlerEvent (p. 85) for more information.

ICnrBeginEditEvent(const IControlEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

ICnrBeginEditEvent

~ICnrBeginEditEvent

```
virtual  
    ~ICnrBeginEditEvent();
```

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

Event Information

Use these members to query and set the accessible attributes of this class.

currentText Returns the current text stored in the edit field.

```
IString  
    currentText();
```

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

Inherited Public Functions

ICnrEditEvent		
column	isLeftDetails	isRightDetailsHeading
container	isLeftDetailsHeading	isTitleWindow
isDetailsData	isRightDetails	object

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

ICnrBeginEditEvent

IBase		
asString	messageText	version

Inherited Protected Functions

ICnrEditEvent		
textRef	textSize	

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrControlList

ICnrControlList

Derivation ISet
ICnrControlList

Inherited By None.

Header File icnrclst.hpp

Members	Member	Page
	Constructor	60
	~ICnrControlList	60

The ICnrControlList class defines a collection of IContainerControl objects. The User Interface Class Library uses this class in the implementation of the class IContainerControl (p. 174).

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

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrControlList

1	ICnrControlList();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

This is the default constructor, which accepts no parameters.

2	ICnrControlList(const ICnrControlList&);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

You can construct an object of this class from a pre-existing ICnrControlList object.

~ICnrControlList

ICnrControlList

```
virtual  
  ~ICnrControlList();
```

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

ICnrDate

ICnrDate

Derivation Inherits from none.

Inherited By None.

Header File idate.hpp

Members	Member	Page
	day	62
	month	62
	year	62

The ICnrDate structure is used to create date column objects for the IContainerControl.

Public Data

Structure Definition

ICnrDate structure definition.

day Day.

unsigned char
day;

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

month Month.

unsigned char
month;

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

year Year.

unsigned short
year;

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



ICnrDrawBackgroundEvent

ICnrDrawBackgroundEvent

Derivation IBase
 IVBase
 IEvent
 ICnrDrawBackgroundEvent

Inherited By None.

Header File icnrdiv.hpp

Members	Member	Page	Member	Page
	Constructor	64	itemPresSpaceHandle	64
	container	64	itemRect	64
	itemId	64	~ICnrDrawBackgroundEvent	64

The ICnrDrawBackgroundEvent class provides the information necessary to draw the background of the container. This event is not dispatched unless one of the following occurs:

- The IContainerControl::handleDrawBackground style is set when the container is created.
- The IContainerControl::enableDrawBackground function is called after construction.

See IContainerControl::handleDrawBackground (p. 227) for information about the handleDrawBackground style and IContainerControl::enableDrawBackground (p. 200) for information about that function.



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of the container background. This event is not used in this case.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrDrawBackgroundEvent

ICnrDrawBackgroundEvent

You can only construct objects of this class from an IEvent object. See IEvent (Vol. II) for information about that class.

```
ICnrDrawBackgroundEvent( const IEvent& event );
```

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

~ICnrDrawBackgroundEvent

```
virtual  
~ICnrDrawBackgroundEvent();
```

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

Event Information

Use these members to query and set the accessible attributes of this class.

container Returns the container being drawn.

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

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

itemId Returns the item's ID.

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

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

itemPresSpaceHandle

Returns the item's presentation space handle.

```
virtual IPresSpaceHandle  
itemPresSpaceHandle() const;
```

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

itemRect Returns the rectangle of the item.

```
virtual IRectangle  
itemRect() const;
```

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

Inherited Public Functions

ICnrDrawBackgroundEvent

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icnr dih d .hpp

Members				
Member	Page	Member	Page	
Constructor	70	drawTitle	69	
dispatchHandlerEvent	70	drawTreeIcon	69	
drawBackground	69	handleEventsFor	68	
drawDetailsItem	69	stopHandlingEventsFor	68	
drawIcon	69	~ICnrDrawHandler	68	
drawText	69			

The ICnrDrawHandler class provides the notification of draw item and draw background events in the container.

To dispatch draw item events, you must do either of the following:

- Specify the IContainerControl::handleDrawItem attribute when you construct the container.
- Call IContainerControl::enableDrawItem after you construct the container.

Similarly, to dispatch draw background events, you must do either of the following:

- Specify the IContainerControl::handleDrawBackground attribute when you construct the container.
- Call IContainerControl::enableDrawBackground after you construct the container.

See IContainerControl::Attributes (p. 223) for information about the handleDrawItem and handleDrawBackground container control attributes. Also, see IContainerControl::enableDrawItem (p. 200) and IContainerControl::enableDrawBackground (p. 200) for information about those functions.

ICnrDrawHandler

In addition, whenever you draw a container item or background, call `IEvent::setResult(true)` to prevent the system from drawing the item or background. See `IEvent::setResult` (Vol. II) for information about that function.

Create a handler derived from `ICnrDrawHandler` and attach it to the container whose foreground or background is to be drawn or to the container's owner window. You can do this by calling `ICnrDrawHandler::handleEventsFor` to pass the container or owner window to the container draw handler. See `ICnrDrawHandler::handleEventsFor` (p. 68) for information about that function.

`ICnrDrawHandler` dispatches the following events in the container:

- Drawing the container background (see `ICnrDrawBackgroundEvent`) (p. 63)
- Drawing a details view item, an icon, text, a title, or a tree view icon (see `ICnrDrawItemEvent`) (p. 72)

Note: Attaching `ICnrDrawHandler` to a container's owner window does not allow you to process `ICnrDrawBackgroundEvents`. Only `ICnrDrawItemEvents` are processed if `ICnrDrawHandler` is attached to an owner window.

When the container draw handler receives one of these container draw events, the handler creates a corresponding event object and routes that object to the appropriate `ICnrDrawHandler` virtual function. You must override these virtual functions to supply your own specialized processing of a container draw event.

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

Return Value	Meaning
true	The container draw event requires no additional processing. Do not pass it to another handler.
false	Pass the container draw event to the next handler for additional processing, as follows: <ul style="list-style-type: none">• If there is another handler for the container, pass the container draw event to the next handler.• If this is the last handler for the container, call <code>IWindow::defaultProcedure</code> to process <code>ICnrDrawBackgroundEvents</code> and call <code>IWindow::dispatch</code> to dispatch <code>ICnrDrawItemEvents</code> to the container's owner window. See <code>IWindow::defaultProcedure</code> (Vol. II) and <code>IWindow::dispatch</code> (Vol. II) for information about those functions.

ICnrDrawHandler

- If this is the last handler for the container's owner window, call `IWindow::defaultProcedure` to process the `ICnrDrawItemEvents`. The handler should not receive any `ICnrDrawBackgroundEvents`.



The native Windows containers (that is, containers constructed without the `pmCompatible` style) do not support owner drawing. This handler is not called in this case.

Public Functions

Constructors

You can destruct objects of this class. The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected.

`~ICnrDrawHandler`

```
virtual  
~ICnrDrawHandler();
```

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

Event Dispatching

Use these members to implement the class.

`handleEventsFor`

Attaches the handler to the specified container control.

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

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

`stopHandlingEventsFor`

Detaches the handler from the specified container control.

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

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

Event Processing

Event-processing members must be supplied by a derived class to provide processing for a draw event.

ICnrDrawHandler

drawBackground

Called when the background of the container needs to be redrawn.

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

drawDetailsItem

Called in the details view when the data of an object in a column needs to be redrawn. The emphasis of the data must be drawn during the processing of this event. This is also called when the details view column title needs to be redrawn. In this case, a call to ICnrDrawItemEvent::object() returns 0.

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

drawIcon

Called when the icon in the icon or name view needs to be redrawn. The emphasis of the icon must be drawn during the processing of this event.

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

drawText

Called when the text in the icon, name, or text view needs to be redrawn. The emphasis of the text must be drawn during the processing of this event.

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

drawTitle

Called when the title of the container needs to be redrawn.

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

drawTreeIcon

Called in the tree view when the expanded or collapsed icon needs to be redrawn.

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

ICnrDrawHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can destruct objects of this class. The only way to construct objects of this class is from a derived class. To enforce this, the only constructor we provide for this class is protected.

ICnrDrawHandler

Constructs instances of this class by using the default constructor, which does not accept any arguments.

ICnrDrawHandler();

Win

PM

Motif

Y

Y

I

Event Dispatching

Use these members to implement the class.

dispatchHandlerEvent

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

virtual Boolean
dispatchHandlerEvent(IEvent& event);

Win

PM

Motif

Y

Y

I

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrDrawItemEvent

ICnrDrawItemEvent

Derivation

IBase
IVBase
IEvent
IControlEvent
IDrawItemEvent
ICnrDrawItemEvent

Inherited By None.

Header File icnrdiv.hpp

Members				
	Member	Page	Member	Page
	Constructor	72	itemRect	73
	column	73	itemType	73
	container	73	object	73
	itemPresSpaceHandle	73	~ICnrDrawItemEvent	73

The ICnrDrawItemEvent class provides the information necessary to draw objects in the container. This event is not dispatched unless one of the following occurs:

- The IContainerControl::handleDrawItem style is set when the container is created.
- The IContainerControl::enableDrawItem function is called after construction.

See Style (p. 225) for information about the handleDrawItem style and IContainerControl::enableDrawItem (p. 200) for information about that function.



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects. This event is not used in this case.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrDrawItemEvent

You can only construct objects of this class from an IEvent object. See IEvent (Vol. II) for information about that class.

ICnrDrawItemEvent

ICnrDrawItemEvent(const IEvent& event);

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

~ICnrDrawItemEvent

virtual
~ICnrDrawItemEvent();

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

Event Information

Use these members to query and set the accessible attributes of this class.

column If an object or column title is being drawn in the details view, the column is returned. If the view is other than details view, 0 is returned.

virtual IContainerColumn*
column() const;

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

container Returns the container being drawn.

virtual IContainerControl*
container() const;

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

itemPresSpaceHandle

Returns the item's presentation space.

virtual IPresSpaceHandle
itemPresSpaceHandle() const;

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

itemRect Returns the rectangle of the item's size.

virtual IRectangle
itemRect() const;

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

itemType Returns the type of the item being drawn.

virtual ItemType
itemType() const;

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

object Returns the object in the container being drawn. If the current view is details view and the column titles are being drawn, then 0 is returned.

ICnrDrawItemEvent

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

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

Inherited Public Functions

IDrawItemEvent		
itemId	itemPresSpaceHandle	itemRect

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDrawItemEvent		
ownerItemData		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ItemType

ItemType {
 text, icon, treeIcon, detailsView, title
};

Enumeration of the possible styles providing the type of the item being drawn:

- text**
The text in all views except the details view.
- icon**
An icon in the icon, name or tree view.
- treeIcon**
An expanded or collapsed tree icon.
- detailsView**
Any drawing in the details view.
- title**
The container title.



Inherited By ICnrBeginEditEvent
ICnrEndEditEvent
ICnrReallocStringEvent

Header File icnreevt.hpp

Members		Member	Page	Member	Page
	Constructor		77	isRightDetailsHeading	78
	column		77	isTitleWindow	78
	container		77	object	79
	isDetailsData		77	textRef	79
	isLeftDetails		77	textSize	80
	isLeftDetailsHeading		78	~ICnrEditEvent	77
	isRightDetails		78		

The ICnrEditEvent class forms the base of the container edit events. These events are dispatched during direct editing in the container. Editing can be started by the user or by program control using the following functions:

IContainerControl::editObject (p. 197)
IContainerControl::editColumnTitle (p. 197)
IContainerControl::editContainerTitle (p. 197)

Direct editing of a field in the container is not allowed unless the field is set to "editable." Whether the container is editable is controlled by the following styles:

IContainerControl::Style::readOnly (p. 225)
IContainerColumn::Style::readOnly (p. 168)
IContainerColumn::Style::readOnlyHeading (p. 168)

and the following functions:

IContainerObject::enableDataUpdate (p. 263)
IContainerObject::disableDataUpdate (p. 263)

ICnrEditEvent

The ICnrEditEvent class and its derived classes provide information describing the location in the container being edited as well as describing the contents of the data being edited.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEditEvent

You can only construct objects of this class from an IControlEvent object. See IControlEvent (Vol. II) for information about that class.

ICnrEditEvent(const IControlEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~ICnrEditEvent

virtual ~ICnrEditEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Event Information

Use these members to query and set the accessible attributes of this class.

column Returns the column being edited. If the edit is not column-related (for example, a title edit), 0 is returned.

virtual IContainerColumn* column() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

container Returns the container in which the edit is taking place.

virtual IContainerControl* container() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

isDetailsData If data in a column of the details view is being edited, true is returned.

Boolean isDetailsData() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

isLeftDetails If data in the left column of the details view is being edited, true is returned.

ICnrEditEvent

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isLeftDetails() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar. There is no right details window in this case.

isLeftDetailsHeading

If the left details heading is being edited, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isLeftDetailsHeading() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of column headings.

isRightDetails

If data in the right column of the details view is being edited, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isRightDetails() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar. There is no right details window in this case.

isRightDetailsHeading

If the right details heading is being edited, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isRightDetailsHeading() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of column headings.

isTitleWindow

If the title is being edited, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTitleWindow() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of container title.

ICnrEditEvent

object Returns the object being edited. If the edit is not object-related (for example, a title edit), 0 is returned.

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

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

Inherited Public Functions

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Text

These functions get information about the text string.

textRef Returns a pointer to the current text pointer for the ICnrBeginEditEvent and ICnrReallocStringEvent events. For ICnrEndEditEvent, a pointer to the new text pointer is returned.

ICnrEditEvent

See ICnrBeginEditEvent (p. 57), ICnrReallocStringEvent (p. 116) and ICnrEndEditEvent (p. 90) for information about those classes.

char **
textRef() const;

Win
Y

PM
Y

Motif
Y

textSize Returns the number of bytes in the text string (not counting the NULL termination) for ICnrEndEditEvent and ICnrReallocStringEvent events. Otherwise, 0 is returned.

See ICnrEndEditEvent (p. 90) and ICnrReallocStringEvent (p. 116) for information about those classes.

unsigned long
textSize() const;

Win
Y

PM
Y

Motif
Y

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrEditHandler

Derivation

- IBase
- IVBase
- IHandler
- ICnrEditHandler

Inherited By None.

Header File icnrehdr.hpp

Members				
Member	Page	Member	Page	
Constructor	82	mleHandler	84	
beginEdit	83	multiLineEdit	84	
dispatchHandlerEvent	85	reallocateString	84	
endEdit	83	setMLEHandler	84	
handleEventsFor	83	stopHandlingEventsFor	83	
isDataString	83	~ICnrEditHandler	82	

The ICnrEditHandler class supports direct editing of data in the container.

Create a handler derived from ICnrEditHandler and attach it to the container whose data is to be edited. You can do this by calling ICnrEditHandler::handleEventsFor (p. 83) to pass the container or the container's owner window to the container edit handler.

ICnrEditHandler dispatches the following events in the container:

- Begin direct editing (see ICnrBeginEditEvent) (p. 57)
- End direct editing (see ICnrEndEditEvent) (p. 90)
- Reallocate storage for edited data (see ICnrReallocStringEvent) (p. 116)

When the container edit handler receives one of these container edit events, it creates a corresponding event object and routes that object to the appropriate ICnrEditHandler virtual function. You must override these virtual functions to supply your own specialized processing of a container edit event.

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

ICnrEditHandler

Return Value	Meaning
true	The container edit event requires no additional processing. Do not pass it to another handler.
false	Pass the container edit event to the next handler for additional processing, as follows: <ul style="list-style-type: none">• If there is another handler for the container, pass the container edit event to the next handler.• If this is the last handler for the container, call <code>IWindow::dispatch</code> (Vol. II) to dispatch the container edit event to the container's owner window.• If this is the last handler for the owner window, call <code>IWindow::defaultProcedure</code> (Vol. II) to process the container edit event.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEditHandler

You can only construct instances of this class by providing the type of the user data that needs to be edited. The data can be one of the types specified by the `StringType` (p. 85) enumeration.

The default behavior of this class provides for the reallocation of the user data. The default behavior of this class also manages the allocation of other types of editable data, such as the following:

- Icon text
- Container title
- Details view column headings

```
ICnrEditHandler( StringType stringType = isIString );
```

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

~ICnrEditHandler

```
virtual  
~ICnrEditHandler();
```

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

ICnrEditHandler

Event Dispatching

Use these members to implement the class.

handleEventsFor

Attaches the handler to the specified container control.

<pre>virtual ICnrEditHandler& handleEventsFor(IContainerControl* container);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

stopHandlingEventsFor

Detaches the handler from the specified container control.

<pre>virtual ICnrEditHandler& stopHandlingEventsFor(IContainerControl* container);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

Event Processing

Event-processing members must be supplied by a derived class to provide processing for an edit event.

beginEdit

Called when an edit window has been opened in the container. The default behavior creates an IMultiLineEdit (Vol. II) object by calling multiLineEdit (p. 84). If setMLEHandler (p. 84) has been previously called to store an edit handler, the handler is added to the multiline entry field control.

<pre>virtual Boolean beginEdit(ICnrBeginEditEvent& event);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

endEdit

Called when editing has ended. This is received after reallocateString (p. 84) is called.

<pre>virtual Boolean endEdit(ICnrEndEditEvent& event);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

isDataIString If the user data is type IString (Vol. I), true is returned.

<pre>Boolean isDataIString() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

ICnrEditHandler

reallocString

Called when text has been modified in the container and storage needs to be reallocated. If the data is an IString (Vol. I), the reallocation is provided by calling ICnrReallocStringEvent::reallocString (p. 117). If the data is of the type isCharacterPointer, the reallocation is provided by calling ICnrReallocStringEvent::reallocText (p. 117).

```
virtual Boolean  
    reallocString( ICnrReallocStringEvent& event );
```

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

Multiline Edit Processing

Use these members to enable processing of events to the container's edit field.

mleHandler Called to return the handler for a multiline entry field. If a handler has not been provided, 0 is returned.

```
IHandler*  
    mleHandler() const;
```

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

multiLineEdit Called to return an IMultiLineEdit (Vol. II) "wrapper." You can override this function to specialize the edit control.

```
virtual IMultiLineEdit*  
    multiLineEdit( const IWindowHandle& handleMultiLineEdit );
```

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

setMLEHandler

Stores a handler to be added to the multiline entry field control when that control is created.

```
virtual void  
    setMLEHandler( IHandler* anMLEHandler );
```

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

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

ICnrEditHandler

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Use these members to implement the class.

dispatchHandlerEvent

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

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

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

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	

StringType StringType {
 isIString,
 isCharacterPointer
 };

Enumeration identifying the type of user data in the details view of the container:

ICnrEditHandler

isIString

The data is an IString (Vol. I) object.

isCharacterPointer

The data is a pointer to a character (char*).



ICnrEmphasisEvent

Derivation

```

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrEmphasisEvent
  
```

Inherited By None.

Header File icnrevt.hpp

Members	Member	Page	Member	Page
	Constructor	87	object	88
	changed	88	~ICnrEmphasisEvent	88
	isAcquired	88		

The ICnrEmphasisEvent class retrieves information about a container emphasis event. Objects of this 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.


Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEmphasisEvent

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (p. 102) creates objects of this class from an object of the class IControlEvent (Vol. II) in response to a change-in-emphasis notification from the container.

	ICnrEmphasisEvent(const IControlEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

ICnrEmphasisEvent

2

ICnrEmphasisEvent(IControlEvent& event,
 IContainerObject* object,
 Boolean acquiring);

Win

PM

Motif

N

N

Y

~ICnrEmphasisEvent

virtual

~ICnrEmphasisEvent();

Win

PM

Motif

Y

Y

Y

Event Information

Use these members to query and set the accessible attributes of this class.

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

Boolean

changed(IContainerObject::Emphasis emphasis) const;

Win

PM

Motif

Y

Y

Y

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

Boolean

isAcquired() const;

Win

PM

Motif

N

N

Y

object Returns the object that has changed its emphasis.

IContainerObject*

object() const;

Win

PM

Motif

Y

Y

Y

Inherited Public Functions

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter 2	setHandle
eventId	passToOwner	setPassToOwner

ICnrEmphasisEvent

IEvent		
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrEndEditEvent

ICnrEndEditEvent

Derivation

Inherited By None.

Header File icnreevt.hpp

Members	Member	Page	Member	Page
	Constructor	90	newText	91
	currentText	91	~ICnrEndEditEvent	90

The ICnrEndEditEvent class retrieves information about a container direct edit event. Objects of this class are dispatched when text editing is complete.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEndEditEvent

You can only construct objects of this class from an IControlEvent (Vol. II) object. Construct ICnrEndEditEvent by calling ICnrEditHandler::dispatchHandlerEvent (p. 85) from an IControlEvent in response to a notification from the container that direct editing has ended. Such an occurrence is either as a result of user interaction or under program control.

```
ICnrEndEditEvent( const IControlEvent& evt );
```

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

~ICnrEndEditEvent

ICnrEndEditEvent

```
virtual  
    ~ICnrEndEditEvent();
```

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

Event Information

Use these members to query and set the accessible attributes of this class.

currentText Returns the current text stored in the container.

If ICnrReallocStringEvent::reallocateString (p. 117) returns true, indicating that the text was reallocated, the returned text is the text entered by the user in the edit field.

If ICnrReallocStringEvent::reallocateString returns false, indicating that the text was not reallocated, the returned text is the text that was in the edit field before editing began.

```
IStrng  
    currentText();
```

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

newText Returns the text the user entered in the edit field.

```
IStrng  
    newText();
```

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

Inherited Public Functions

ICnrEditEvent		
column	isLeftDetails	isRightDetailsHeading
container	isLeftDetailsHeading	isTitleWindow
isDetailsData	isRightDetails	object

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner

ICnrEndEditEvent

IEvent		
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEditEvent		
textRef	textSize	

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrEnterEvent

Derivation

```

IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrEnterEvent
  
```

Inherited By None.

Header File icnrevt.hpp

Members	Member	Page	Member	Page
	Constructor	93	validObject	94
	enterPressed	94	~ICnrEnterEvent	94
	object	94		

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

- The Enter key is pressed.
- A double-click of the mouse select button occurred.

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

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEnterEvent

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (p. 102) creates objects of this class from an object of the class IControlEvent (Vol. II) in response to the user pressing the Enter key or double-clicking the mouse select button.

```
ICnrEnterEvent( const IControlEvent& event );
```

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

ICnrEnterEvent

~ICnrEnterEvent

virtual

~ICnrEnterEvent();

Win

PM

Motif

Y

Y

Y

Event Information

Use these members to query and set the accessible attributes of this class.

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

Boolean

enterPressed() const;

Win

PM

Motif

Y

Y

Y

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

IContainerObject*

object() const;

Win

PM

Motif

Y

Y

Y

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

Boolean

validObject() const;

Win

PM

Motif

Y

Y

Y

Inherited Public Functions

IControlEvent		
controlId		

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

ICnrEnterEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By

ICnrEditEvent
ICnrEmphasisEvent
ICnrEnterEvent

ICnrHelpEvent
ICnrQueryDeltaEvent
ICnrScrollEvent

Header File

icnrevt.hpp

Members

Member	Page
Constructor	96
containerId	97
~ICnrEvent	96

ICnrEvent is the base class for all container events classes.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrEvent

Although you can construct objects of this class, typically `ICnrHandler::dispatchHandlerEvent` (p. 102) creates objects of this class from an object of the class `IControlEvent` (Vol. II) in response to the user pressing the Enter key or double-clicking the mouse select button.

```
ICnrEvent( const IControlEvent& event );
```

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

~ICnrEvent

```
virtual
    ~ICnrEvent();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
γ	γ	γ

Inherited Public Functions

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Information

These members implement the class.

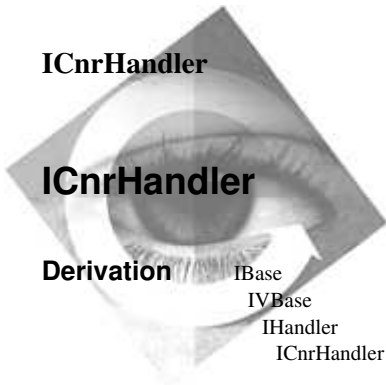
containerId Returns the control ID for the container.

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

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

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icnrhdr.hpp

Members				
Member	Page	Member	Page	
Constructor	99	inuseChanged	101	
cursorChanged	100	selectedChanged	101	
deltaReached	100	stopHandlingEventsFor	99	
dispatchHandlerEvent	102	treeCollapsed	101	
enter	100	treeExpanded	101	
handleEventsFor	99	windowScrolled	101	
help	100	~ICnrHandler	99	

The ICnrHandler class provides the 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` (p. 99) 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` (p. 87).
- Scrolling past the container's delta value. This is described in `ICnrQueryDeltaEvent` (p. 113).
- Responding to an enter event using the keyboard or mouse. This is described in `ICnrEnterEvent` (p. 93).
- Requesting help within the container. This is described in `ICnrHelpEvent` (p. 103).
- Expanding and collapsing nodes in a tree view. This is described in `IContainerObject` (p. 258).
- Scrolling the container window. This is described in `ICnrScrollEvent` (p. 120).

ICnrHandler

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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrHandler Provides the default constructor, which does not accept any parameters.

ICnrHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~ICnrHandler

virtual ~ICnrHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Event Dispatching

Use these members to implement the class.

handleEventsFor

Attaches the handler to the specified container control.

virtual ICnrHandler& handleEventsFor(IContainerControl* container);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

stopHandlingEventsFor

Detaches the handler from the specified container control.

ICnrHandler

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

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

Event Processing

Event-processing members must be supplied by a derived class to provide processing for an edit event.

cursoredChanged

Called when the cursored emphasis of an object changes. By default, this function calls `IContainerObject::handleCursoredChange` (p. 260) on the object whose emphasis has changed.

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

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

deltaReached

Called when the container is scrolled to the predefined delta.

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

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



The native Windows containers (that is, containers constructed without the `pmCompatible` style) do not support data caching. This function is not called in this case.

enter

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` (p. 261). Otherwise, this function ignores the enter.

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

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

help

Called when help is requested in a container.

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

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



By default, this function does nothing.

ICnrHandler

inuseChanged

Called when the in-use emphasis of an object changes. By default, this function calls IContainerObject::handleInuseChange (p. 261).

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

selectedChanged

Called when the selection emphasis of an object changes. By default, this function calls IContainerObject::handleSelectedChange (p. 261).

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
selectedChanged(ICnrEmphasisEvent& event);	<i>Y</i>	<i>Y</i>	<i>Y</i>

treeCollapsed

Called when the specified object is collapsed in the specified container. By default, this function calls IContainerObject::handleTreeCollapse (p. 261).

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
treeCollapsed(IContainerObject* collapsedObject, IContainerControl* container);	<i>Y</i>	<i>Y</i>	<i>Y</i>

treeExpanded

Called when the specified object is expanded in the specified container. By default, this function calls IContainerObject::handleTreeExpand (p. 261) on the expanded object.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
treeExpanded(IContainerObject* expandedObject, IContainerControl* container);	<i>Y</i>	<i>Y</i>	<i>Y</i>

windowScrolled

Called when the container window is scrolled.

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



By default, this function does nothing.

ICnrHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Use these members to implement the class.

dispatchHandlerEvent

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

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

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

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrHelpEvent

Derivation

```

IBase
IVBase
IEvent
IControlEvent
ICnrEvent
ICnrHelpEvent
    
```

Inherited By None.

Header File icnrevt.hpp

Members	Member	Page	Member	Page
	Constructor	103	object	104
	column	104	~ICnrHelpEvent	103

The ICnrHelpEvent class retrieves information about a container help event. 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrHelpEvent

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (p. 102) creates objects of this class from an object of the class IControlEvent (Vol. II) in response to a notification from the container that the user has requested help.

```
ICnrHelpEvent( const IControlEvent& event );
```

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

~ICnrHelpEvent

ICnrHelpEvent

```
virtual
    ~ICnrHelpEvent();
```

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

Event Information

Use these members to query and set the accessible attributes of this class.

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

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

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

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

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

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

Inherited Public Functions

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icnrmhdr.hpp

Members				
Member	Page	Member	Page	
Constructor	108	popupMenuObject	107	
addSourceEmphasis	108	removeSourceEmphasis	109	
dispatchHandlerEvent	109	stopHandlingEventsFor	107	
handleEventsFor	107	~ICnrMenuHandler	107	
menuEnded	109			

The **ICnrMenuHandler** class extends the class **IMenuHandler** (Vol. II) 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** (p. 107) to pass the appropriate container to the container menu handler.

When the container menu handler receives a container menu event, it creates an **IMenuEvent** (Vol. II) 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

~ICnrMenuHandler

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

Event Dispatching

These members are overridden to implement the class.

handleEventsFor

Attaches the handler to the specified container control.

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

virtual ICnrMenuHandler& handleEventsFor(IContainerControl* container);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

stopHandlingEventsFor

Detaches the handler from the specified container control.

virtual ICnrMenuHandler& stopHandlingEventsFor(IContainerControl* container);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Event Information

Use these members to query and set the accessible attributes of this class.

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.

IContainerObject* popupMenuObject();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

ICnrMenuHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Constructors

You can construct and destruct objects of this class.

ICnrMenuHandler

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

```
ICnrMenuHandler();
```

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

Event Dispatching

These members are overridden to implement the class.

addSourceEmphasis

Adds source emphasis by calling `IContainerControl::showSourceEmphasis` (p. 218).

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

Source emphasis is added as follows:

- 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.

ICnrMenuHandler

- If a pop-up menu is selected off of an object, source emphasis is displayed on the container itself.

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

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



This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

dispatchHandlerEvent

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

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

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

menuEnded

Resets the pointer to the menu object that is stored when the menu is created and calls IMenuHandler::menuEnded (Vol. II) to complete the processing.

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

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

removeSourceEmphasis

Removes source emphasis by calling IContainerControl::hideSourceEmphasis (p. 217).

This function reverses the action of addSourceEmphasis.

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

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

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



This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

Inherited Protected Functions

IMenuHandler		
addSourceEmphasis	makePopUpMenu	menuSelected
dispatchHandlerEvent	menuEnded	menuShowing

IHandler		
defaultProcedure	dispatchHandlerEvent	

ICnrMenuHandler

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrObjectSet

Derivation ISequence
 ICnrObjectSet

Inherited By None.

Header File icnrolst.hpp

Members	Member	Page
	Constructor	111
	~ICnrObjectSet	112

The ICnrObjectSet class defines a set of IContainerObjects (p. 258).
 IContainerControl (p. 174) provides several functions that return sets of
 IContainerObjects.

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

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrObjectSet

1	ICnrObjectSet();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

This provides the default constructor, which accepts no parameters.

2	ICnrObjectSet(const ICnrObjectSet&);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

You can construct an object of this class from a pre-existing ICnrObjectSet object.

ICnrObjectSet

~ICnrObjectSet

```
virtual
~ICnrObjectSet();
```

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



ICnrQueryDeltaEvent

Derivation

```


IBase
  IVBase
    IEvent
      IControlEvent
        ICnrEvent
          ICnrQueryDeltaEvent
  
```

Inherited By None.

Header File icnrevt.hpp

Members	Member	Page	Member	Page
	Constructor	113	atLastObject	114
	atBottomDelta	114	atTopDelta	114
	atFirstObject	114	~ICnrQueryDeltaEvent	114

The ICnrQueryDeltaEvent class retrieves information about a container delta event. Objects of this 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 preset delta value has been reached.

 The native Windows containers (that is, containers constructed without the pmCompatible style) do not support data caching. This event is not used in this case.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrQueryDeltaEvent

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (p. 102) creates objects of this class from an object of the class IControlEvent (Vol. II) in response to a notification from the container that the preset delta value has been reached.

```
ICnrQueryDeltaEvent( const IControlEvent& event );
```

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

ICnrQueryDeltaEvent

~ICnrQueryDeltaEvent

```
virtual  
    ~ICnrQueryDeltaEvent();
```

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

Event Information

Use these members to query and set the accessible attributes of this class.

atBottomDelta

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

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

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

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

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

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

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

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

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

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

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

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

Inherited Public Functions

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter l	setEventType

ICnrQueryDeltaEvent

IEvent		
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrReallocStringEvent

ICnrReallocStringEvent



Inherited By None.

Header File icnreevt.hpp

Members	Member	Page	Member	Page
	Constructor	116	reallocateString	117
	currentText	117	reallocateText	117
	newText	117	~ICnrReallocStringEvent	117
	newTextSize	117		

The ICnrReallocStringEvent class retrieves information about a container direct edit event. Objects of this class are dispatched when text is edited in the container and it is necessary to reallocate the storage for the text.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrReallocStringEvent

Constructs objects only from an IControlEvent (Vol. II) object. Construct ICnrReallocStringEvent by calling ICnrEditHandler::dispatchHandlerEvent (p. 85) from an IControlEvent in response to a notification from the container that text has changed and storage for the text may need to be reallocated.

```
ICnrReallocStringEvent( const IControlEvent& event );
```

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

ICnrReallocStringEvent

~ICnrReallocStringEvent

virtual ~ICnrReallocStringEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Event Information

Use these members to query and set the accessible attributes of this class.

currentText Returns the current text stored in the edit field.

virtual IString currentText() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

newText Returns the text the user added to the edit control.

virtual IString newText() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

newTextSize Returns the size (not counting the NULL termination) of the storage that needs to be allocated to store the new text string.

long newTextSize() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Text Allocation

Use these members reallocate the storage for the text field.

reallocateString

Creates and stores an IString (Vol. I) of the appropriate size to hold the new data.

void reallocateString();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

reallocateText

Calls global operator new to acquire storage of the appropriate size to hold the new data.

void reallocateText();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

ICnrReallocStringEvent

Inherited Public Functions

ICnrEditEvent		
column	isLeftDetails	isRightDetailsHeading
container	isLeftDetailsHeading	isTitleWindow
isDetailsData	isRightDetails	object

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter 2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEditEvent		
textRef	textSize	

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icnrevt.hpp

Members	Member	Page	Member	Page
	Constructor	120	isRightDetails	121
	amount	121	isVertical	121
	isHorizontal	121	~ICnrScrollEvent	121
	isLeftDetails	121		

The ICnrScrollEvent class retrieves information about a container scroll event. Objects of this 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICnrScrollEvent

Although you can construct objects of this class, typically ICnrHandler::dispatchHandlerEvent (p. 102) creates objects of this class from an object of the class IControlEvent (Vol. II) in response to a notification from the container that one of the container's windows has been scrolled.

ICnrScrollEvent(const IControlEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

ICnrScrollEvent

~ICnrScrollEvent

virtual			
~ICnrScrollEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

Event Information

Use these members to query and set the accessible attributes of this class.

amount Returns the amount the window has scrolled, in pixels.

long			
amount() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

isHorizontal If scrolling in the horizontal direction, true is returned.

Boolean			
isHorizontal() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

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

Boolean			
isLeftDetails() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

isRightDetails

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

Boolean			
isRightDetails() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

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

Boolean			
isVertical() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

Inherited Public Functions

IControlEvent		
controlId		

ICnrScrollEvent

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

ICnrEvent		
containerId		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICnrTime

Derivation Inherits from none.

Inherited By None.

Header File itime.hpp

Members	Member	Page	Member	Page
	hours	123	seconds	123
	minutes	123	ucReserved	123

The ICnrTime structure is used to create time column objects for the IContainerControl.

Public Data

Structure Definition


ICnrTime structure definition.

hours	Hours.			
	unsigned char hours;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y
minutes	Minutes.			
	unsigned char minutes;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y
seconds	Seconds.			
	unsigned char seconds;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y
ucReserved	Reserved.			
	unsigned char ucReserved;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

ICollectionViewComboBox

ICollectionViewComboBox

Derivation



```
graph TD
    IBase[IBase] --> IVBase[IVBase]
    IVBase --> INotifier[INotifier]
    INotifier --> IWindow[IWindow]
    IWindow --> IControl[IControl]
    IControl --> ITextControl[ITextControl]
    ITextControl --> IEntryField[IEntryField]
    IEntryField --> IBaseComboBox[IBaseComboBox]
    IBaseComboBox --> ICollectionViewComboBox[ICollectionViewComboBox]
```

Inherited By None.

Header File icombovw.hpp

Members

Member	Page	Member	Page
Constructor	128	itemsId	134
collectionReplaced	126	noSelection	134
deselect	130	select	130
elementAdded	127	selectedCollectionPosition	130
elementChanged	127	selectedElement	130
elementDeleted	127	setItems	126
elementsChanged	128	setStringGenerator	131
itemChangedId	134	stringGenerator	131
items	126	~ICollectionViewComboBox	129

The ICollectionViewComboBox<Element, Collection> template class provides a view of an ordered collection, as items in the list box portion of a combination box. Template class Element is the type of objects in the collection, and template class Collection is the type of collection. The sequence of elements is the same between the ordered collection and list box portion of the combination box. The ICollectionViewComboBox interface consists of the following:

- Population and query members for accessing the collection
- The collection-reporting protocol for receiving and handling collection updates
- The string generator for translating elements to IStrings
- Selection members
- Notification members

The setItems member populates the list box portion of the combination box from collection elements. The items member returns the collection being viewed.

ICollectionViewComboBox

The collection-reporting protocol refers to the virtual member functions, such as `elementsChanged`, that handle reported changes in the observed collection. The `ICollectionViewComboBox` object observes the currently viewed collection (initialized on `setItems`) and reports collection updates via calls to the collection-reporting members. Then, the collection-reporting members update the list box portion of the combination box.

The string generator object translates collection elements to their `IString` representations. The default string generator assumes the collection elements natively support operator-> or are `Object*` objects, where `Object` has an `asString` member function. The collection-reporting members use this string-generation mechanism in their processing.

The selection members query and manipulate the selection state of the list box portion of the combination box based on collection elements. This includes the following:

- Selecting or deselecting based on an item's collection position
- Querying what collection element corresponds to the item selected in the list box portion of the combination box

An `ICollectionViewComboBox` object sends notifications, so observers of the object can monitor the following events and process as needed:

- The viewed collection is replaced.
- An item changes in the list box portion of the combination box (due to an update of the corresponding collection element).

The `ICollectionViewComboBox` class design supports use primarily by the Visual Builder. It establishes a direct link between a collection and its visual representation in a list control. If using this class outside the Visual Builder pay particular attention to the following design requirements:

- The collection elements must be pointers.
- The collection element class must derive from `IStandardNotifier`.
- You must use the collection classes enabled for the Visual Builder. These collection templates have an `IV` prefix and are defined in corresponding `ivxxx.h` header files.
- Since the collection elements are pointers, if you require a string generator, use `IStringGeneratorRefMemberFn` objects.

Public Functions

Collection

Use these members to set and retrieve the collection currently being viewed by the `ICollectionViewComboBox` object.

ICollectionViewComboBox

items Returns the collection currently being viewed. Use it to access the underlying collection.

<pre>virtual Collection* items() const;</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setItems Changes the underlying viewed collection for the combination box. Effectively, this repopulates the list box portion of the combination box with items corresponding to the elements in the new collection. This is the only direct way to populate the list box portion of an ICollectionViewComboBox object. All other changes to items in the list box portion of the combination box result from changes to the underlying collection elements reported via the collection-reporting protocol.

collection A pointer to a collection. ICollectionViewComboBox supports collections that are descendents of IPartOrderedCollection.

<pre>virtual ICollectionViewComboBox < Element , Collection >& setItems(Collection* collection);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Collection-Reporting Protocol

Use these members to handle reported changes in the underlying ordered collection. As a set, they are a protocol through which changes in the collection are reported and subsequently processed via updates to the control, in this case, the list box portion of the combination box. ICollectionViewListBox uses this same protocol.

Note: The ordered collection is enabled for notification.

collectionReplaced

Indicates that a new collection is being observed. Effectively, a call to this member notifies an ICollectionViewComboBox object that all items in the list box portion of the combination box were replaced. The member's default behavior sends the ICollectionViewComboBox<Element, Collection>::itemsId notification to all observers of this ICollectionViewComboBox object.

Note: The reporting protocol calls this member after updating all items in the list box portion of the combination box. The elementsChanged member handles the actual updates to the list box items.

<pre>virtual ICollectionViewComboBox < Element , Collection >& collectionReplaced();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

ICollectionViewComboBox

elementAdded

Indicates that an element was added to the collection at the specified position. Effectively, this member notifies ICollectionViewComboBox that an item must be added to the list box portion of the combination box. The default behavior generates a string for the collection element, using the IStringGenerator object, and inserts the generated string into the list box.

position A 1-based index reference to the added collection element.
element The new collection element.

<pre>virtual ICollectionViewComboBox < Element , Collection >& elementAdded(unsigned long position, const Element& element);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

elementChanged

Indicates that a collection element changed state. The collection element's collection position is the first parameter. Effectively, this member notifies an ICollectionViewComboBox object that the list box item corresponding to the collection element needs updating. The default behavior regenerates the string representation for this collection element and replaces the corresponding item in the list box portion of the combination box.

position A 1-based index reference to the changed collection element.
element The collection element that changed.

<pre>virtual ICollectionViewComboBox < Element , Collection >& elementChanged(unsigned long position, const Element& element);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

elementDeleted

Indicates that a collection element was deleted at the specified position. Effectively, this member notifies an ICollectionViewComboBox object that the corresponding item must be deleted from the list box portion of the combination box. The default behavior deletes the list box item.

position A 1-based index that specifies the collection position of the deleted collection element prior to the delete.

<pre>virtual ICollectionViewComboBox < Element , Collection >& elementDeleted(unsigned long position);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

ICollectionViewComboBox

elementsChanged

Indicates that at least one element and possibly all elements in the viewed collection have changed. Effectively, this member notifies an ICollectionViewComboBox object that all items in the list box portion of the combination box need updating based on the current state of the viewed collection. The default behavior deletes all items from the list box portion of the combination box and repopulates that list box from the collection's contents.

<pre>virtual ICollectionViewComboBox < Element , Collection >& elementsChanged();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

ICollectionViewComboBox

1	<pre>ICollectionViewComboBox(unsigned long identifier, IWindow* parent, IWindow* owner, const IRectangle& initial = IRectangle (), const IBaseComboBox::Style& style = IBaseComboBox::defaultStyle (), const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct objects of this class by using the parent window, owner window, optional size and location, optional style, and optional string generator parameters.

<i>id</i>	A combination box control ID.
<i>parent</i>	The parent window.
<i>owner</i>	The owner window.
<i>initial</i>	A rectangle for the control. It specifies the initial position and size of the object you are constructing. Optional.
<i>style</i>	The initial style for the control. The default is IBaseComboBox::classDefaultStyle (Vol. II). Optional.
<i>stringGenerator</i>	The initial string generator for the collection view. Optional.

This creates the specified combination box control and an object for it.

ICollectionViewComboBox

2	<code>ICollectionViewComboBox(unsigned long identifier, IWindow* parent, const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

You can construct objects of this class by using the parent window and optional string generator parameters.

id A combination box control ID.

parent The parent window.

stringGenerator

The initial string generator for the collection view. Optional.

3	<code>ICollectionViewComboBox(const IWindowHandle& handle, const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

You can construct objects of this class by using the handle of an existing combination box and optional string generator parameters.

handle The window handle of an existing combination box control.

stringGenerator

The initial string generator for the collection view. Optional

~ICollectionViewComboBox

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

Selection

Use these members to retrieve and manipulate the selection state of an ICollectionViewComboBox object. These members act on the list box portion of the combination box but use the collection as their reference for parameters and return values. The parameters and return values are collection positions or collection elements. The selection actions include the following:

- Selecting or deselecting an item
- Retrieving the selected item

Note: A collection operates as if it is a 1-based array of items. The collection position is such an index.

ICollectionViewComboBox

deselect

Deselects the item in the list box portion of the combination box corresponding to the collection element at collectionPosition.

collectionPosition

A 1-based index reference to a collection element.

Note: In collections, the collection position is a 1-based index (the first item is item 1).

virtual ICollectionViewComboBox < Element , Collection >& deselect(unsigned long collectionPosition);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

select

Selects or deselects the item in the list box portion of the combination box corresponding to the collection element at collectionPosition.

collectionPosition

A 1-based index reference to a collection element.

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.

Note: In collections, the collection position is a 1-based index (the first item is item 1).

virtual ICollectionViewComboBox < Element , Collection >& select(unsigned long collectionPosition, Boolean select = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

selectedCollectionPosition

Returns the collection position (1-based index) corresponding to the selected item in the list box portion of the combination box.

Note: If no item is selected, noSelection is returned.

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

selectedElement

Returns the collection element corresponding to the selected item in the list box portion of the combination box.

Note: If no item is selected, an exception is thrown.

virtual Element selectedElement() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

ICollectionViewComboBox

Exceptions	
InvalidRequest	There is no selected list box item.

String Generator

Use these members to set and retrieve the string generator associated with an ICollectionViewComboBox object. The default string generator uses Element asString to return a string. This member assumes that Element natively supports operator-> or is an Object* object, where Object has an asString member function.

setStringGenerator

Replaces the string generator associated with an ICollectionViewComboBox object. The string generator provides strings for collection elements for use as items in the list box portion of the combination box. Typically, a string generator contains a IStringGeneratorMemberFn, which is an Element member function. The string generator generates strings by calling the contained member function. Thus, use this member to replace the Element member function used to produce strings for the collection view items.

stringGenerator

A string generator.

<pre>virtual IStringGenerator < Element >& setStringGenerator(const IStringGenerator < Element >& stringGenerator);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

stringGenerator

Retrieves the string generator associated with an ICollectionViewComboBox object. The string generator provides strings for collection elements for use in the list box part of the combination box. Use this function to assign one IStringGenerator to another; this increments the use count of the optionally contained IStringGeneratorFn.

<pre>virtual IStringGenerator < Element >& stringGenerator();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

Inherited Public Functions

IBaseComboBox		
convertToGUIStyle	itemHandle	setDefaultStyle
count	itemText	setForegroundColor

ICollectionViewComboBox

IBaseComboBox		
defaultStyle	layoutAdjustment	setItemHandle
deselect	limit	setItemText
deselectAll	locateText	setLimit
elementAt	minimumRows	setMinimumRows
enableNotification	moveSizeTo	setTop
hasFocus	nativeRect	showList
hideList	numberOfSelections	size
isEmpty	position	top
isHorizontalScroll	select	topHandle
isListShowing	selection	type
isSelected	setBackgroundColor	visibleRectangle

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

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

ICollectionViewComboBox

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

IBaseComboBox		
calcMinimumSize	incrementChangeCount	setLayoutDistorted
changeCount	registerCallbacks	unregisterCallbacks

IEntryField		
calcMinimumSize	isDragStarting	registerCallbacks
initialize	passEventToOwner	setLayoutDistorted

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

These members define the possible notifications that ICollectionViewComboBox provides to its observers. The following events can be observed:

ICollectionViewComboBox

- An item changed in the list box portion of the combination box.
- All items in the list box portion of the combination box changed.

itemChangedId

Notification identifier provided to observers when a combination box item changes due to a change in the underlying collection element.

ICollectionViewCombinationBox<Element, Collection> provides the changed collection element in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
itemChangedId;	<i>Y</i>	<i>Y</i>	<i>N</i>

itemsId

Notification identifier provided to observers when the collection underlying the collection view combination box is changed. ICollectionViewCombinationBox<Element, Collection> provides a pointer to the new collection in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
itemsId;	<i>Y</i>	<i>Y</i>	<i>N</i>

Selection

Use these members to retrieve and manipulate the selection state of an ICollectionViewComboBox object. These members act on the list box portion of the combination box but use the collection as their reference for parameters and return values. The parameters and return values are collection positions or collection elements. The selection actions include the following:

- Selecting or deselecting an item
- Retrieving the selected item

Note: A collection operates as if it is a 1-based array of items. The collection position is such an index.

noSelection

Indicates no item is selected in the list box portion of the combination box. ICollectionViewCombinationBox::selectedCollectionPosition returns this value.

static const unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noSelection;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

IBaseComboBox		
anyData	dropDownType	notFound
autoScroll	enterId	oemData
border3D	first	readOnlyDropDownType
classDefaultStyle	horizontalScroll	sbsData
dbcsData	mixedData	selectId

IEntryField		
anyData	command	lowerCase
autoScroll	dataUpdateId	margin
autoTab	dbcsData	mixedData
border3D	end	readOnly
centerAlign	insertModeId	rightAlign
characterTypeId	leftAlign	sbsData
classDefaultStyle	limitId	unreadable

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

ICollectionViewComboBox

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ICollectionViewConstants

Derivation IBase
ICollectionViewConstants

Inherited By None.

Header File icollvwi.hpp

Members	Member	Page	Member	Page
	comboDelete	137	listEnd	138
	comboDeleteAll	138	listError	138
	comboInsert	138	listInsert	138
	listDelete	138	listMemoryError	138
	listDeleteAll	138	listNone	139

The ICollectionViewConstants class provides constants for the ICollectionViewListBox and ICollectionViewComboBox template implementation. Thus, the constants are intended for use by the ICollectionViewListBox and ICollectionViewComboBox classes only. So, adhere to the following:

- Do not create objects of this class.
- Do not use these constants.

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Public Data

Template Implementation

You do not use these constants.

comboDelete You do not use this constant.

ICollectionViewConstants

<code>static const unsigned long comboDelete;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

comboDeleteAll

You do not use this constant.

<code>static const unsigned long comboDeleteAll;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

comboInsert You do not use this constant.

<code>static const unsigned long comboInsert;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

listDelete You do not use this constant.

<code>static const unsigned long listDelete;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

listDeleteAll You do not use this constant.

<code>static const unsigned long listDeleteAll;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

listEnd You do not use this constant.

<code>static const unsigned long listEnd;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

listError You do not use this constant.

<code>static const unsigned long listError;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

listInsert You do not use this constant.

<code>static const unsigned long listInsert;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

listMemoryError

You do not use this constant.

ICollectionViewConstants

```
static const unsigned long  
listMemoryError;
```

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

listNone You do not use this constant.

```
static const unsigned long  
listNone;
```

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

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ICollectionViewListBox

ICollectionViewListBox

Derivation

```
graph TD; IBase --> IVBase; IVBase --> INotifier; INotifier --> IWindow; IWindow --> IControl; IControl --> IBaseListBox; IBaseListBox --> ICollectionViewListBox;
```

Inherited By None.

Header File ilistcvw.hpp

Members

Member	Page	Member	Page
Constructor	144	itemsId	150
collectionReplaced	142	noSelection	151
deselect	145	select	146
elementAdded	143	selectedCollectionPosition	146
elementChanged	143	selectedElement	147
elementDeleted	143	selectedElements	147
elementsChanged	143	setItems	142
enableExtendedSelect	146	setStringGenerator	148
extendedSelectChangedId	150	stringGenerator	148
itemChangedId	150	~ICollectionViewListBox	145
items	142		

The `ICollectionViewListBox<Element, Collection>` template class provides a view of an ordered collection, as items in a list box. Template class `Element` is the type of objects in the collection, and template class `Collection` is the type of collection. The sequence of elements is the same between the ordered collection and list box. The `ICollectionViewListBox` interface consists of the following:

- Population and query members for accessing the collection
- The collection-reporting protocol for receiving and handling collection updates
- The string generator for translating elements to `IStrings`
- Selection members
- Notification members

The `setItems` member populates the list box from collection elements. The `items` member returns the collection being viewed.

The collection-reporting protocol refers to the virtual member functions, such as `elementsChanged`, that handle reported changes in the observed collection.

ICollectionViewListBox

ICollectionViewListBox objects observe the currently viewed collection (initialized on setItems) and report collection updates via calls to the collection-reporting members. Then, the collection-reporting members update the list box.

The string generator object translates collection elements to their IString representations. The default string generator assumes that the collection elements natively support operator-> or are Object* objects, where Object has an asString member function. The collection-reporting members use this string generation mechanism in their processing.

The selection members query and manipulate the selection state of the list box based on collection elements. This includes the following:

- Selecting or deselecting based on an item's collection position
- Querying what collection elements correspond to the selected items in the list box

An ICollectionViewListBox object sends notifications so that observers can monitor, and process the following events:

- The viewed collection is replaced.
- The extended selection style attribute changes.
- An item changes in the list box (due to an update of the corresponding collection element).

The ICollectionViewListBox class design supports use primarily by the Visual Builder. It establishes a direct link between a collection and its visual representation in a list control. If using this class outside the Visual Builder pay particular attention to the following design requirements:

- The collection elements must be pointers.
- The collection element class must derive from IStandardNotifier.
- You must use the collection classes enabled for the Visual Builder. These collection templates have an IV prefix and are defined in corresponding ivxxx.h header files.
- Since the collection elements are pointers, if you require a string generator, use IStringGeneratorRefMemberFn objects.

Public Functions

Collection

Use these members to set and retrieve the collection currently being viewed by the ICollectionViewListBox object.

ICollectionViewListBox

items Returns the collection currently being viewed. Typically, items is used by the collection-reporting members to retrieve the collection that this control is responsible for viewing.

<code>virtual Collection* items() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setItems Changes the underlying viewed collection for the list box. Effectively, this repopulates the list box with items corresponding to the elements in the new collection. This is the only direct way to populate the ICollectionViewListBox object's list box. All other changes to list box items result from changes to the underlying collection elements reported via the collection-reporting protocol.

collection A pointer to a collection. ICollectionViewListBox supports collections that are descendents of IPartOrderedCollection.

<code>virtual ICollectionViewListBox < Element , Collection > & setItems(Collection* collection);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Collection-Reporting Protocol

Use these members to handle reported changes in the underlying ordered collection. As a set, they are a protocol through which changes in the collection are reported and subsequently processed via updates to the control, in this case, the list box. ICollectionViewComboBox uses this same protocol.

Note: The ordered collection is enabled for notification.

collectionReplaced

Indicates that a new collection is being observed. Effectively, a call to this member notifies an ICollectionViewListBox object that all items in the list box were replaced. The member's default behavior sends the ICollectionViewListBox<Element, Collection>::itemsId notification to all observers of this ICollectionViewListBox object.

Note: The collection-reporting protocol calls this member after updating all items in the list box. The elementsChanged member handles the actual updates to the list box items.

<code>virtual ICollectionViewListBox < Element , Collection > & collectionReplaced();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

ICollectionViewListBox

elementAdded

Indicates that an element was added to the collection at the specified position. Effectively, this member notifies ICollectionViewListBox that an item must be added to the list box. The default behavior generates a string for the collection element, using the IStringGenerator object, and inserts the generated string into the list box.

position A 1-based index reference to the added collection element.
element The new collection element.

<pre>virtual ICollectionViewListBox < Element , Collection >& elementAdded(unsigned long position, const Element& element);</pre>	<table><tbody><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

elementChanged

Indicates that a collection element at the specified position changed its state. Effectively, this member notifies an ICollectionViewListBox object that the list box item corresponding to the collection element needs updating. The default behavior regenerates the string representation for this collection element and replaces the corresponding item in the list box.

position A 1-based index reference to the changed collection element.
element The collection element that changed.

<pre>virtual ICollectionViewListBox < Element , Collection >& elementChanged(unsigned long position, const Element& element);</pre>	<table><tbody><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

elementDeleted

Indicates that a collection element was deleted at the specified position. Effectively, this member notifies an ICollectionViewListBox object that the corresponding item must be deleted from the list box. The default behavior deletes the list box item.

position A 1-based index that specifies the collection position of the deleted collection element prior to the delete.

<pre>virtual ICollectionViewListBox < Element , Collection >& elementDeleted(unsigned long position);</pre>	<table><tbody><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

elementsChanged

Indicates that at least one element and possibly all elements in the viewed collection have changed. Effectively, this member notifies an ICollectionViewListBox object that all items in the list box need updating based on the current state of the viewed

ICollectionViewListBox

collection. The default behavior deletes all items from the list box and repopulates that list box from the collection's contents.

<code>virtual ICollectionViewListBox < Element , Collection >& elementsChanged();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Constructors

You can construct and destruct objects of this class.

ICollectionViewListBox

1	<code>ICollectionViewListBox(unsigned long identifier, IWindow* parent, IWindow* owner, const IRectangle& initial = IRectangle (), const IBaseListBox::Style& style = IBaseListBox::defaultStyle (), const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

You can construct objects of this class by using the parent window, owner window, optional size and location, optional style, and optional string generator parameters.

<i>id</i>	A list box control ID.
<i>parent</i>	The parent window.
<i>owner</i>	The owner window.
<i>initial</i>	A rectangle for the control. It specifies the initial position and size of the object you are constructing. Optional.
<i>style</i>	The initial style for the control. The default is <code>IBaseListBox::classDefaultStyle</code> (Vol. II). Optional.
<i>stringGenerator</i>	The initial string generator for the collection view. Optional.

This creates the specified list box control and an object for it.

2	<code>ICollectionViewListBox(unsigned long identifier, IWindow* parent, const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

ICollectionViewListBox

You can construct objects of this class by using the parent window and optional string generator parameters.

id A list box control ID.

parent The parent window.

stringGenerator
 The initial string generator for the collection view. Optional.

3	<pre>ICollectionViewListBox(const IWindowHandle& handle, const IStringGenerator < Element >& stringGenerator = IStringGenerator < Element > ());</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

You can construct objects of this class by using the handle of an existing list box and optional string generator parameters.

handle The window handle of an existing list box control.

stringGenerator
 The initial string generator for the collection view. Optional.

~ICollectionViewListBox

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

Selection

Use these members to retrieve and manipulate the selection state of an ICollectionViewListBox object. These members act on the list box but use the collection as their reference for parameters and return values. The parameters and return values are collection positions or collection elements. The selection actions include the following:

- Selecting or deselecting an item
- Retrieving the selected item
- Retrieving the selected items as a collection

Note: A collection operates as if it is a 1-based array of items. The collection position is such an index.

deselect Deselects the item in the list box corresponding to the collection element at collectionPosition.

ICollectionViewListBox

collectionPosition

A 1-based index reference to a collection element.

Note: In collections, the collection position is a 1-based index (the first item is item 1).

```
virtual ICollectionViewListBox < Element , Collection >&
    deselect( unsigned long collectionPosition );
```

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

enableExtendedSelect

Enables or disables the style IBaseListBox::extendedSelect (Vol. II) for the list box and sends the IBaseCollectionListBox<Element, Collection>::extendedSelectChangedId notification.

```
virtual ICollectionViewListBox < Element , Collection >&
    enableExtendedSelect( Boolean extended = true );
```

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

select

Selects or deselects the item in the list box corresponding to the collection element at collectionPosition.

collectionPosition

A 1-based index reference to a collection element.

select If you specify true, based on the list box's selection style, the following occurs:

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

Multiple selection The specified item is selected.

Extended selection The specified item is selected.

The default is true.

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

Note: In collections, the collection position is a 1-based index (the first item is item 1).

```
virtual ICollectionViewListBox < Element , Collection >&
    select( unsigned long collectionPosition,
           Boolean select = true );
```

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

selectedCollectionPosition

Returns the collection position (1-based index) corresponding to the selected item in the list box portion of the combination box. For single-selection list boxes, the index

ICollectionViewListBox

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, noSelection is returned.

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

selectedElement

Returns the collection element corresponding to the selected item in the list box. For single-selection list boxes, the element corresponding to the selected item is returned. For multiple-selection or extended-selection list boxes, the element corresponding to the *first* selected item is returned.

Note: If no item is selected, an exception is thrown.

virtual Element selectedElement() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Exceptions	
InvalidRequest	There are no list box items selected.

selectedElements

Returns a collection of elements corresponding to the selected list box items. It deletes all items from the collection parameter before filling it with the elements corresponding to the selected list box items. Thus, if no items are selected, an empty collection is returned.

elements A collection for returning elements corresponding to the selected list box items.

virtual ICollectionViewListBox < Element , Collection >& selectedElements(Collection& elements);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

String Generator

Use these members to set and retrieve the string generator associated with the ICollectionViewListBox object. The default string generator uses Element asString to return a string. This member assumes that Element natively supports operator-> or is an Object* object, where Object has an asString member function.

ICollectionViewListBox

setStringGenerator

Replaces the string generator associated with an ICollectionViewListBox object. The string generator provides strings for collection elements for use as items in the list box portion of the combination box. Typically, a string generator contains a IStringGeneratorMemberFn, which is an Element member function. The string generator generates strings by calling the contained member function. Thus, use this member to replace the Element member function used to produce strings for the collection view items.

stringGenerator

A string generator.

```
virtual IStringGenerator < Element >&
    setStringGenerator(
        const IStringGenerator < Element >& stringGenerator);
```

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

stringGenerator

Retrieves the string generator associated with an ICollectionViewListBox object. The string generator provides strings for collection elements for use as list box items.

Note: IStringGenerator supports copying or assigning IStringGenerator objects; it handles the proper reference-counting of the optionally contained IStringGeneratorFn.

```
virtual IStringGenerator < Element >&
    stringGenerator();
```

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

Inherited Public Functions

IBaseListBox		
backgroundColor	enableNoAdjustPosition	minimumRows
convertToGUIStyle	enableNotification	numberOfSelections
count	isDrawItem	select
defaultStyle	isEmpty	selectAll
deselect	isExtendedSelect	selection
deselectAll	isHorizontalScroll	setDefaultStyle
disableDrawItem	isMultipleSelect	setItemHandle
disableExtendedSelect	isNoAdjustPosition	setItemHeight
disableMultipleSelect	isSelected	setItemText

ICollectionViewListBox

IBaseListBox		
disableNoAdjustPosition	itemHandle	setLayoutDistorted
elementAt	itemHeight	setMinimumCharacters
enableDrawItem	itemText	setMinimumRows
enableExtendedSelect	locateText	setTop
enableMultipleSelect	minimumCharacters	show

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

IBaseListBox		
calcMinimumSize	incrementChangeCount	registerCallbacks
changeCount	passEventToOwner	unregisterCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

ICollectionViewListBox

Public Data

Notification Members

These members define the possible notifications that ICollectionViewListBox provides to its observers. The following events can be observed:

- An item changed in the list box.
- All items in the list box changed.
- The extended selection style for the list box changed.

extendedSelectChangedId

Notification identifier provided to observers when the extended select state of the list box changes.

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
extendedSelectChangedId;	<i>Y</i>	<i>Y</i>	<i>N</i>

itemChangedId

Notification identifier provided to observers when a list box item changes due to a change in the underlying collection element. ICollectionViewListBox<Element, Collection> provides the changed collection element in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
itemChangedId;	<i>Y</i>	<i>Y</i>	<i>N</i>

itemsId

Notification identifier provided to observers when the collection underlying the collection view list box is changed. ICollectionViewListBox<Element, Collection> provides the a pointer to the new collection in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
itemsId;	<i>Y</i>	<i>Y</i>	<i>N</i>

Selection

Use these members to retrieve and manipulate the selection state of an ICollectionViewListBox object. These members act on the list box but use the collection as their reference for parameters and return values. The parameters and return values are collection positions or collection elements. The selection actions include the following:

- Selecting or deselecting an item

ICollectionViewListBox

- Retrieving the selected item
- Retrieving the selected items as a collection

Note: A collection operates as if it is a 1-based array of items. The collection position is such an index.

noSelection Indicates no list box item is selected.
ICollectionViewListBox::selectedCollectionPosition returns this value.

```
static const unsigned long  
noSelection;
```

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

Inherited Public Data

IBaseListBox		
border3D	enterId	horizontalScroll
classDefaultStyle	extendedSelect	multipleSelect
drawItem	first	noAdjustPosition

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

ICollectionViewListBox

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerColumn

Derivation IBase
IVBase
IContainerColumn

Inherited By None.

Header File icnrcol.hpp

Members	Member	Page	Member	Page
	Constructor	155	invisible	169
	alignBottom	171	isDate	161
	alignCentered	171	isHeadingIconHandle	162
	alignLeft	171	isHeadingString	162
	alignRight	171	isHeadingWriteable	162
	alignTop	171	isIconHandle	162
	alignVerticallyCentered	171	isNumber	162
	classDefaultDataStyle	168	isString	162
	classDefaultHeadingStyle	170	isTime	163
	columnInfo	167	isVisible	163
	container	167	isWriteable	163
	dataAsDate	156	justifyData	163
	dataAsIcon	156	justifyHeading	163
	dataAsNumber	156	number	169
	dataAsString	157	readOnly	169
	dataAsTime	157	readOnlyHeading	170
	dataAttributes	167	setColumnInfo	167
	date	168	setContainer	167
	defaultDataStyle	157	setDataAttributes	168
	defaultHeadingStyle	158	setDataOffset	164
	disableDataUpdate	158	setDefaultDataStyle	157
	disableHeadingUpdate	158	setDefaultHeadingStyle	158
	displayWidth	158	setDisplayWidth	164
	enableDataUpdate	159	setHeadingIcon	164
	enableHeadingUpdate	159	setHeadingText	165
	handleDrawItem	169	setHelpId	165
	hasHorizontalSeparator	159	setTitleAttributes	168
	hasVerticalSeparator	160	show	165
	headingIcon	160	showSeparators	166
	headingText	160	string	172
	helpId	160	time	170
	hide	160	titleAttributes	168
	hideSeparators	161	verticalDataAlignment	166
	horizontalDataAlignment	161	verticalHeadingAlignment	166
	horizontalHeadingAlignment	161	verticalSeparator	170
	horizontalSeparator	169	~IContainerColumn	156
	icon	171		
	invalidate	167		

IContainerColumn

The IContainerColumn class provides 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 (p. 258) 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 noneditable, left-justified, and centered vertically

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



The native Windows containers (that is, containers constructed without the pmCompatible style) have the following restrictions for columns:

- Vertical column separators are not supported.
- Horizontal column separators are required.
- Heading styles and alignment are defined by DataStyle.
- Headings must be text and are read-only.
- Split bar is not supported.
- Only the first column may have an icon.

IContainerColumn

- The first column is required. It must be the icon and text that is defined in the IContainerObject and displayed in the nondetails views. This column is initially sized to a zero width and is made nonzero if a column that is constructed with DataSource=isIcon or DataSource=isIconViewText is added. Regardless of where you attempt to add these columns, they will be at the beginning.
- The required column is the only column that displays selection emphasis. If this column has a zero width, then no selection emphasis will be displayed.
- The required column's heading is defined by the heading text of the column constructed with DataSource=isIconViewText if it is inserted.



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 represents 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 leftmost column uses the headingText set for that column.

IContainerControl::ColumnCursor (p. 236) ignores the automatically created leftmost column.

Public Functions

Constructors

You can construct and destruct objects of this class.

IContainerColumn

1	<code>IContainerColumn(unsigned long dataOffset, const HeadingStyle& title = defaultHeadingStyle (), const DataStyle& data = defaultDataStyle ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Construct an IContainerColumn object 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.

2	<code>IContainerColumn(DataSource objectDataType, const HeadingStyle& title = defaultHeadingStyle (), const DataStyle& data = defaultDataStyle ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Construct an IContainerColumn object by providing a type of data that is part of IContainerObject (p. 258) (as opposed to the user portion of the object) and, optionally, the styles to use for the heading and data.

IContainerColumn



The native Windows containers (that is, containers constructed without the pmCompatible style) require that the first column be the icon text pair that is present in the other views. Because of this, regardless of where you attempt to insert a column constructed with this constructor, it will be placed at the beginning. If you insert more than one of the same type, then the second one will be invalid and not visible.

	<code>IContainerColumn(const IContainerColumn& column);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

Construct an IContainerColumn object by copying from an existing container column.

~IContainerColumn

<code>virtual ~IContainerColumn();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Data Retrieval

Use these members to retrieve data that is stored in an object referenced by this class.

dataAsDate Returns the data referenced by this column in the specified object. The data is returned as an IDate (Vol. I). 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.

<code>IDate dataAsDate(const IContainerObject* object) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

dataAsIcon Returns the data referenced by this column in the specified object. The data is returned as an IPointerHandle (Vol. II). 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.

<code>IPointerHandle dataAsIcon(const IContainerObject* object) const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

dataAsNumber

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.

IContainerColumn

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
dataAsNumber(const IContainerObject* object) const;	Y	Y	Y

dataAsString 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.

IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
dataAsString(const IContainerObject* object) const;	Y	Y	Y

dataAsTime Returns the data referenced by this column in the specified object. The data is returned as an ITime (Vol. I). If the type of the data is not IContainerColumn::time (p. 170), an IInvalidRequest (Vol. I) exception is thrown. You can call IContainerColumn::isTime to determine if this function is valid for a column object.

ITime	<u>Win</u>	<u>PM</u>	<u>Motif</u>
dataAsTime(const IContainerObject* object) const;	Y	Y	Y

Data Style

In addition to the IContainerColumn::Style types, the data in a column can also contain these data styles.

defaultDataStyle

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

static Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultDataStyle();	Y	Y	Y

setDefaultDataStyle

Sets the default data style for all subsequent column data.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultDataStyle(const DataStyle& dataStyle);	Y	Y	Y

Heading Style

In addition to the IContainerColumn::Style types, the heading can also contain these styles.

IContainerColumn

defaultHeadingStyle

Returns the default heading style. The default heading style is `classDefaultHeadingStyle` (p. 170) unless you have changed it using `setDefaultHeadingStyle` (p. 158).

<code>static Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>defaultHeadingStyle();</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

setDefaultHeadingStyle

Sets the default heading style for all subsequent column headings.

<code>static void</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>setDefaultHeadingStyle(const HeadingStyle& headingStyle);</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the `pmCompatible` style) column heading style is defined by the columns data style.

Object Information

Use these members to query and set the accessible attributes of this class.

disableDataUpdate

Prevents editing of the column data.

<code>virtual IContainerColumn&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>disableDataUpdate();</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

disableHeadingUpdate

Prevents editing of the heading.

<code>virtual IContainerColumn&</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>disableHeadingUpdate();</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the `pmCompatible` style) do not support editing of column headings.

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 as follows:

IContainerColumn

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

This rectangle's width is either of the following:

- 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.

virtual unsigned long displayWidth();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------

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

enableDataUpdate

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

virtual IContainerColumn& enableDataUpdate(Boolean enable = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

enableHeadingUpdate

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

virtual IContainerColumn& enableHeadingUpdate(Boolean enable = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of column headings.

hasHorizontalSeparator

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

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

Win The native Windows containers (that is, containers constructed without the pmCompatible style) require a horizontal separator for columns.

IContainerColumn

hasVerticalSeparator

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

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasVerticalSeparator() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support vertical separators for columns.

headingIcon

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

virtual IPointerHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
headingIcon() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support icons for column headings.

headingText

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

virtual IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
headingText() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

helpId

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 displays this help panel when the user requests help while direct-editing the column data of the details view.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
helpId() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>



The AIX release does not support this attribute because AIX does not support the ICnrDrawHandler.

hide

Hides the column by making it invisible.

virtual IContainerColumn&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hide();	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of columns.

IContainerColumn

hideSeparators

Removes one of the following:

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

By default, this function removes both separators.

<pre>virtual IContainerColumn& hideSeparators(const DataStyle& separatorStyles = horizontalSeparator verticalSeparator);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support vertical separators for columns and require horizontal separators.

horizontalDataAlignment

Returns the current horizontal alignment of the column data in a container details view. The returned value is an enumerator provided by HorizontalAlignment (p. 172).

<pre>HorizontalAlignment horizontalDataAlignment() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

horizontalHeadingAlignment

Returns the current horizontal alignment of the column heading data in a container details view. The returned value is an enumerator provided by HorizontalAlignment (p. 172).

<pre>HorizontalAlignment horizontalHeadingAlignment() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) column heading alignment is defined by the columns data alignment.

isDate

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

<pre>Boolean isDate() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

IContainerColumn

isHeadingIconHandle

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

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isHeadingIconHandle() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support icons for column headings.

isHeadingString

If the data in the column heading is a character string, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isHeadingString() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) only support strings for column headings.

isHeadingWriteable

If the data in the column heading can be edited, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isHeadingWriteable() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of column headings. This function always returns false in this case.

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

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isIconHandle() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isNumber If the data in the column is a number, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isNumber() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isString If the data in the column is a pointer to a character string or an IString (Vol. I), true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isString() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerColumn

isTime If the data in the column is a time, true is returned.

Boolean
isTime() const;

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

isVisible If the column is visible in the container, true is returned.

Boolean
isVisible() const;

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of columns. All invalid columns inserted will be made not visible by default. This function will return false for these invalid columns.

isWriteable If the user can edit the data in the column, true is returned.

Boolean
isWriteable() const;

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

justifyData Sets the justification of the data to a horizontal and vertical alignment. Use the enumeration HorizontalAlignment (p. 172) to set the horizontal alignment to left, right, or centered. VerticalAlignment (p. 173) sets the vertical alignment to top, bottom, or centeredVertically.

virtual IContainerColumn&
justifyData(VerticalAlignment = centeredVertically,
HorizontalAlignment = centered);

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

justifyHeading

Sets the justification of the heading to a horizontal and vertical alignment. Use the enumeration HorizontalAlignment (p. 172) to set the horizontal alignment to left, right, or centered. VerticalAlignment (p. 173) sets the vertical alignment to top, bottom, or centeredVertically.

virtual IContainerColumn&
justifyHeading(VerticalAlignment = centeredVertically,
HorizontalAlignment = centered);

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



The native Windows containers (that is, containers constructed without the pmCompatible style) column heading alignment is defined by the column's data alignment.

IContainerColumn

setDataOffset

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 (p. 258) 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.

<pre>virtual IContainerColumn& setDataOffset(unsigned long dataOffset);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setDisplayWidth

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 as follows:

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

This rectangle's width is either of the following:

- 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.

<pre>virtual IContainerColumn& setDisplayWidth(unsigned long widthInPixels);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

setHeadingIcon

Sets the icon to use for the heading and changes the style to icon.

IContainerColumn

1	virtual IContainerColumn& setHeadingIcon(unsigned long iconId);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>I</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support icons for column headings.

2	virtual IContainerColumn& setHeadingIcon(const IPointerHandle& iconHandle);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>I</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support icons for column headings.

3	virtual IContainerColumn& setHeadingIcon(const IResourceId& iconId);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>I</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support icons for column headings.

setHeadingText

Sets the text to use for the heading and changes the style to string.

1	virtual IContainerColumn& setHeadingText(const IResourceId& textId);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

2	virtual IContainerColumn& setHeadingText(const char* text);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

setHelpId

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.

virtual IContainerColumn& setHelpId(unsigned long helpId);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

show

Makes the column visible.

virtual IContainerColumn& show(Boolean visible = true);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of columns.

IContainerColumn

showSeparators

- Adds any of the following:
- The vertical separator after a column
 - The horizontal separator under the column heading
 - Both of the preceding separators

By default, this function adds both separators.

Note: This function removes any separators previously set. If the column currently has a horizontal separator and showSeparators(verticalSeparator) is used to add a vertical separator, the horizontal separator is removed.

```
virtual IContainerColumn&
showSeparators(
    const DataStyle& separatorStyles =
        horizontalSeparator | verticalSeparator);
```

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

Win

The native Windows containers (that is, containers constructed without the pmCompatible style) do not support vertical separators for columns and require horizontal separators.

verticalDataAlignment

Returns the current alignment of column data in a container details view. The returned value is an enumerator provided by VerticalAlignment (p. 173).

```
VerticalAlignment
verticalDataAlignment() const;
```

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

verticalHeadingAlignment

Returns the current alignment of column heading data in a container details view. The returned value is an enumerator provided by VerticalAlignment (p. 173).

```
VerticalAlignment
verticalHeadingAlignment() const;
```

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

Win

The native Windows containers (that is, containers constructed without the pmCompatible style) column heading alignment is defined by the columns data alignment.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IContainerColumn

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Object Attributes

These members implement the class.

columnInfo Returns the address of the container's column record.

<code>_FIELDINFO* columnInfo() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

container Returns the container in which this column is located.

<code>IContainerControl* container() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

dataAttributes

Returns the current data attributes in the container.

<code>virtual unsigned long dataAttributes() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

invalidate If the column is in a container, the column is refreshed.

<code>IContainerColumn& invalidate();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setColumnInfo

Stores the address of the container's column record.

<code>IContainerColumn& setColumnInfo(_FIELDINFO* fieldinfo);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setContainer Stores the specified container, which is the container this column is located in.

IContainerColumn

IContainerColumn& setContainer(IContainerControl* container);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

setDataAttributes

Sets the specified attributes into the data.

virtual IContainerColumn& setDataAttributes(unsigned long dataAttributes);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

setTitleAttributes

Sets the specified attributes into the title.

virtual IContainerColumn& setTitleAttributes(unsigned long titleAttributes);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

titleAttributes

Returns the current title attributes in the container.

virtual unsigned long titleAttributes() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	------------------------	-----------------------	--------------------------

Public Data

Data Style

In addition to the IContainerColumn::Style types, the data in a column can also contain these data styles.

classDefaultDataStyle

Specifies the original default style for data in a column, which is string, alignVerticallyCentered, alignLeft, and readOnly.

static const DataStyle classDefaultDataStyle;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

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:

1. A single byte for the day
2. A single byte for the month
3. Two bytes for the year

IContainerColumn

static const DataStyle
date;

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

PM In the OS/2 operating system, 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 (p. 72) to be dispatched to ICnrDrawHandler::drawDetailsItem (p. 69).

static const DataStyle
handleDrawItem;

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

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects.

horizontalSeparator

Draws a horizontal separator beneath column headings.

static const DataStyle
horizontalSeparator;

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

Win The native Windows containers (that is, containers constructed without the pmCompatible style) provide horizontal separators as a requirement.

invisible

Hides the data in the column by making it invisible.

static const DataStyle
invisible;

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

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of columns.

number

Specifies the data is an unsigned long number.

static const DataStyle
number;

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

readOnly

Disables editing of the data in the column.

static const DataStyle
readOnly;

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

IContainerColumn

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:

1. A single byte for the hours
2. A single byte for the minutes
3. A single byte for the seconds
4. A final byte, which is reserved

```
static const DataStyle  
time;
```

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



In the OS/2 operating system, 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.

```
static const DataStyle  
verticalSeparator;
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support vertical separators for columns.

Heading Style

In addition to the IContainerColumn::Style types, the heading can also contain these styles.

classDefaultHeadingStyle

Specifies the original default style for a heading, which is string, alignVerticallyCentered, alignLeft, and readOnlyHeading.

```
static const HeadingStyle  
classDefaultHeadingStyle;
```

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

readOnlyHeading

Disables editing the data in the heading.

```
static const HeadingStyle  
readOnlyHeading;
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of column headings.

Styles

These style members are both heading styles and data styles. You can specify `alignTop`, `alignBottom`, and `alignVerticallyCentered`, and only one of `alignLeft`, `alignRight`, and `alignCentered`.

alignBottom Aligns the data to the bottom of the cell.

```
static const Style
    alignBottom;
```

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

alignCentered

Centers the data horizontally in the cell.

```
static const Style
    alignCentered;
```

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

alignLeft Aligns the data to the left of the cell.

```
static const Style
    alignLeft;
```

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

alignRight Aligns the data to the right of the cell.

```
static const Style
    alignRight;
```

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

alignTop Aligns the data to the top of the cell.

```
static const Style
    alignTop;
```

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

alignVerticallyCentered

Centers the data vertically in the cell.

```
static const Style
    alignVerticallyCentered;
```

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

icon Specifies that the data is an icon.

```
static const Style
    icon;
```

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

IContainerColumn

string Specifies that the data is a string.

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

Inherited Protected Data

IBase		
<code>recoverable</code>	<code>unrecoverable</code>	

Nested Type Definitions

DataSource `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
 `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.

IContainerColumn

VerticalAlignment

```
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.

HeadingStyle typedef unsigned long HeadingStyle;

A flag used to represent construction values for the column heading.

Style typedef unsigned long Style;

A flag used to represent construction values for both the column heading and the column data.

DataStyle typedef unsigned long DataStyle;

A flag used to represent construction values for the data in a column.

IContainerControl

IContainerControl

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> INotifier
    INotifier --> IWindow
    IWindow --> IControl
    IControl --> IContainerControl
```

Inherited By None.

Header File icnrcctl.hpp

Members

Member	Page	Member	Page
Constructor	191	currentEditColumn	197
addColumn	192	currentEditMLE	197
addId	229	currentEditObject	197
addObject	178	cursorObject	212
addObjectAfter	178	defaultAttribute	187
addObjects	179	defaultStyle	220
addObjectsAfter	179	deleteAllObjects	179
alignTitleCentered	226	deleteSelectedObjects	179
alignTitleLeft	226	descendentsOf	213
alignTitleRight	226	detailsObjectRectangle	205
allObjectsDo	199	detailsTitleRectangle	192
areDetailsViewTitlesVisible	180	detailsView	226
arrangeIconView	188	detailsViewPortOnWindow	210
attributes	223	detailsViewPortOnWorkspace	210
autoPosition	231	detailsViewSplit	193
backgroundColor	206	detailsViewTitles	226
baseRecordSize	224	detailsViewTitlesId	229
calcMinimumSize	224	disableCaching	199
classDefaultAttribute	226	disableDataUpdate	216
classDefaultStyle	231	disableDrawBackground	199
closeEdit	196	disableDrawItem	200
collapse	215	disableDrop	216
collapseTree	220	disableTitleUpdate	181
column	224	editColumnTitle	197
columnAt	192	editContainerTitle	197
columnCount	192	editObject	197
columnUnderPoint	204	enableCaching	200
containerAttributes	224	enableDataUpdate	216
containerFromHandle	199	enableDrawBackground	200
containerKey	224	enableDrawItem	200
containerList	224	enableDrop	216
containsObject	212	enableNotification	204
convertToGUIStyle	219	enableTitleUpdate	181
convertToWorkspace	204	enterId	230
copyObjectTo	202	expand	216

IContainerControl

Member	Page	Member	Page
expandTree	220	isTreeIconView	196
extendedSelection	231	isTreeNameView	196
filter	198	isTreeTextView	196
flowedView	226	isTreeView	196
foregroundColor	207	isVisible	207
handleDrawBackground	227	isWriteable	209
handleDrawItem	227	lineSpacing	184
hasMixedTargetEmphasis	181	miniIcons	227
hasNormalTargetEmphasis	181	mixedTargetEmphasis	228
hasOrderedTargetEmphasis	181	moveIconTo	206
hideDetailsViewTitles	182	moveObjectTo	203
hideObject	216	multipleSelection	231
hideSourceEmphasis	217	nameView	228
hideSplitBar	193	nlsCompare	219
hideTitle	182	noSharedObjects	231
hideTitleSeparator	182	numberOfColumnChanges	201
hideTreeLine	220	numberOfObjectChanges	201
hwndAllocation	225	objectAt	213
iconRectangle	206	objectCount	213
iconSize	182	objectList	213
iconView	227	objectUnderPoint	206
immediateDescendantsOf	213	operator ==	191
initialize	198	orderedTargetEmphasis	228
isCachingEnabled	201	parentObject	213
isCollapsed	208	pmCompatible	232
isColumnRight	193	readOnly	232
isCursored	208	readOnlyTitle	228
isDetailsView	194	refresh	211
isDragStarting	223	refreshAllContainers	212
isDrawBackgroundEnabled	201	registerCallbacks	223
isDrawItemEnabled	201	removeAllObjects	180
isDropOnAble	208	removeColumn	193
isExpanded	208	removeColumnAt	193
isExtendedSelection	182	removeId	230
isFlowed	194	removeInUse	217
isFlowedNameView	195	removeObject	180
isFlowedTextView	195	removeObjectAt	180
isIconView	195	removeSelected	217
isInUse	209	removeSelectedObjects	180
isMoveValid	202	resetBackgroundColor	207
isMultipleSelection	182	resetForegroundColor	207
isNameView	195	scroll	214
isPMCompatible	183	scrollDetailsHorizontally	214
isRefreshOn	211	scrollHorizontally	215
isSelected	209	scrollToObject	215
isShowingMiniIcons	195	scrollVertically	215
isSingleSelection	183	selectId	230
isSource	209	sendEvent	207
isTarget	209	setAttributes	225
isTextView	196	setBackgroundColor	208
isTitleSeparatorVisible	183	setContainerAttributes	225
isTitleVisible	183	setCursor	217
isTitleWriteable	183		

IContainerControl

Member	Page	Member	Page
setDefaultAttribute	187	showNameView	189
setDefaultStyle	220	showObject	218
setDeleteColumnsOnClose	187	showSourceEmphasis	218
setDeleteObjectsOnClose	188	showSplitBar	194
setDetailsViewSplit	194	showTextView	190
setEditColumn	198	showTitle	186
setEditMLE	198	showTitleSeparator	186
setEditObject	198	showTreeIconView	190
setEmphasis	225	showTreeLine	221
setExtendedSelection	184	showTreeNameView	190
setForegroundColor	208	showTreeTextView	190
setIconSize	184	singleSelection	232
setInUse	218	sort	219
setLineSpacing	184	sortByIconText	219
setMixedTargetEmphasis	184	splitBarOffset	194
setMultipleSelection	185	textRectangle	206
setNormalTargetEmphasis	185	textView	228
setOrderedTargetEmphasis	185	title	186
setRefreshOff	212	titleId	230
setRefreshOn	212	titleRectangle	187
setSelected	218	titleSeparator	229
setSingleSelection	185	titleVisibleId	230
setTitle	185	topHandle	222
setTitleAlignment	186	treeView	229
setTreeExpandIconSize	220	unregisterCallbacks	223
setTreeItemIcons	221	verifyPointers	232
setTreeViewIndent	221	viewPortOnWindow	210
showDetailsView	188	viewPortOnWorkspace	210
showDetailsViewTitles	186	visibleTitle	229
showFlowedNameView	189	visibleTreeLine	229
showFlowedTextView	189	willDeleteColumnsOnClose	188
showIconView	189	willDeleteObjectsOnClose	188
showMiniIcons	189	~IContainerControl	191

The IContainerControl class displays 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

IContainerControl

PM A container will not paint over sibling windows that lie over it. For this reason, a container can be hidden by a sibling combination box, group box, or outline box control, even if the control does not paint the portion of its window that overlaps the container (for example, the interior of a group box). You can cause the container to paint correctly by placing it on top of all overlapping sibling windows, either by changing the order that you create it relative to its sibling windows, or by using function `IWindow::positionOnSiblings` (Vol. II) or `IWindow::positionBehindSibling` (Vol. II).

Win The native Windows containers (that is, containers constructed without the `pmCompatible` style) are wrappers of the Windows list view and tree view controls. Both controls are maintained simultaneously by `IContainerControl`. The underlying controls are only created when necessary. If you create the container in a tree view and never switch to a nontree view, then a Windows list view control is never created. This approach is recommended for applications never using a nontree view. Likewise, a container that is created in a nontree view and never switches to a tree view or adds child objects will not cause an underlying Windows tree view control to be created and maintained.

The native Windows containers do have limitations when compared to the `pmCompatible` containers. These limitations are documented in each of the affected functions. The most notable limitations are in the details view and are described in functions and classes used by the details view.

The native Windows containers do not support constructing this control in the following ways:

- Loaded in a dialog template
- As a pre-existing control

Motif 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.

IContainerControl

- Scrolling functions, for example:
 - scroll
 - scrollDetailsHorizontally
 - scrollHorizontally
 - scrollVertically
 - scrollToObject
- The IContainerControl constructors that create an object from an existing container control or a dialog template.
- The ICnrHandler overridden member functions that the library does not call:
 - censoredChanged
 - deltaReached
 - inuseChanged
 - windowScrolled

Public Functions

Adding and Removing Objects

Use these members are used to add and remove objects.

addObject 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.

<pre>virtual IContainerControl& addObject(const IContainerObject* newObject, IContainerObject* parentObject = 0);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td>Y</td><td>Y</td><td>Y</td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

addObjectAfter

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

IContainerControl

If *afterObject* is nonzero, the container ignores *parentObject*. The level, in this case, is determined by *afterObject*.


```
virtual IContainerControl&
    addObjectAfter( const IContainerObject* newObject,
                   const IContainerObject* afterObject,
                   IContainerObject* parentObject = 0 );
```

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

addObjects Adds multiple objects to the container. In tree view, the parent defines the location of the objects.

```
virtual IContainerControl&
    addObjects( ICnrAllocator& allocator,
               IContainerObject* parentObject = 0 );
```

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


 The native Windows containers (that is, containers constructed without the pmCompatible style) do not support adding multiple objects with one call. This function can be used, but the objects will be inserted separately.

addObjectsAfter

Adds multiple objects to be placed after a given object. If you specify 0 for the *afterObject* argument, the objects are added as the first objects in the container.

```
virtual IContainerControl&
    addObjectsAfter( ICnrAllocator& allocator,
                    const IContainerObject* afterObject,
                    IContainerObject* parentObject = 0 );
```

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

 The native Windows containers (that is, containers constructed without the pmCompatible style) do not support adding multiple objects with one call. This function can be used, but the objects will be inserted separately.

deleteAllObjects

Deletes all objects in this container. If an object exists in other containers, it is removed from them as well before being deleted.

```
virtual IContainerControl&
    deleteAllObjects();
```

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

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.

```
virtual IContainerControl&
    deleteSelectedObjects();
```

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

IContainerControl

removeAllObjects

Removes all objects from the container.

<pre>virtual IContainerControl& removeAllObjects();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

removeObject

Removes the specified object from this container or, optionally, all containers.

<pre>virtual IContainerControl& removeObject(IContainerObject* object, Boolean allContainers = false);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

removeObjectAt

Removes the object at the specified cursor's position.

1 <pre>virtual IContainerControl& removeObjectAt(IContainerControl::ObjectCursor& cursor);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

2 <pre>virtual IContainerControl& removeObjectAt(IContainerControl::TextCursor& cursor);</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

removeSelectedObjects

Removes all selected objects from the container.

<pre>virtual IContainerControl& removeSelectedObjects();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

Appearance

Use these members to query and set the accessible attributes of this class.

areDetailsViewTitlesVisible

Queries whether the column titles are currently displayed for details view.

<pre>Boolean areDetailsViewTitlesVisible() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
---	-------------------------------	------------------------------	---------------------------------

IContainerControl



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support hiding of column titles. This function always returns true for these containers.

disableTitleUpdate

Disables the container title from being edited.

```
virtual IContainerControl&  
    disableTitleUpdate();
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the container title.

enableTitleUpdate

Allows the container title to be edited.

```
virtual IContainerControl&  
    enableTitleUpdate( Boolean enable = true );
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the container title.

hasMixedTargetEmphasis

Queries if the mixed target emphasis mode is set for drag operations over this container.

```
Boolean  
    hasMixedTargetEmphasis() const;
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support mixed target emphasis for drag and drop.

hasNormalTargetEmphasis

Queries if the regular target emphasis mode is set for drag operations over this container.

```
Boolean  
    hasNormalTargetEmphasis() const;
```

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

hasOrderedTargetEmphasis

Queries if the ordered target emphasis mode is set for drag operations over this container.

IContainerControl

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasOrderedTargetEmphasis() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support ordered target emphasis for drag and drop.

hideDetailsViewTitles

Removes column headings from details view.

virtual IContainerControl& hideDetailsViewTitles();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support hiding of the column titles.

hideTitle

Removes the container title from the container.

virtual IContainerControl& hideTitle();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

hideTitleSeparator

Removes the title separator from the container window.

virtual IContainerControl& hideTitleSeparator();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

iconSize

Returns the icon or bitmap size of all objects.

ISize iconSize() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

isExtendedSelection

Queries whether the container is in extended-selection mode. *Extended selection* is an enhanced version of single selection that lets the user select discontinuous sets of container items.

Boolean isExtendedSelection() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

isMultipleSelection

Queries whether the container is in multiple-selection mode. *Multiple selection* is a mode that allows the user unrestricted selection of objects.

IContainerControl

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isMultipleSelection() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a multiple selection mode. Extended selection is used instead.

isPMCompatible

Returns true if the pmCompatible style is set for this container.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isPMCompatible() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

PM This function always returns true.

Win This function returns false if the pmCompatible style is not set and the native Windows containers are being used.

isSingleSelection

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.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isSingleSelection() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

isTitleSeparatorVisible

Queries whether the title separator is currently displayed.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTitleSeparatorVisible() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

isTitleVisible Queries whether the container title is currently displayed.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTitleVisible() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

isTitleWritable

Returns true if the data in the title can be edited.

IContainerControl

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTitleWriteable() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the container title.

lineSpacing 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.

long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
lineSpacing() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support changing the spacing between container objects.

setExtendedSelection

Sets the selection mode to extended selection. Extended selection is an enhanced version of single selection that lets the user select discontinuous sets of container items.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setExtendedSelection();	<i>Y</i>	<i>Y</i>	<i>Y</i>

setIconSize Sets the icon or bitmap size for all objects.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setIconSize(const ISize& iconSize);	<i>Y</i>	<i>Y</i>	<i>I</i>

setLineSpacing

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.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setLineSpacing(long lineSpacing);	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support changing the spacing between container objects.

setMixedTargetEmphasis

Sets the drag mode to mixed-target emphasis.

IContainerControl

```
virtual IContainerControl&  
    setMixedTargetEmphasis();
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support mixed-target emphasis for drag and drop.

setMultipleSelection

Sets the selection mode to multiple selection. *Multiple selection* is a mode that gives the user unrestricted selection of objects.

```
virtual IContainerControl&  
    setMultipleSelection();
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support multiple selection mode. Extended selection is used instead.

setNormalTargetEmphasis

Sets the drag mode to normal-target emphasis.

```
virtual IContainerControl&  
    setNormalTargetEmphasis();
```

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

setOrderedTargetEmphasis

Sets the drag mode to ordered-target emphasis.

```
virtual IContainerControl&  
    setOrderedTargetEmphasis();
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support ordered-target emphasis for drag and drop.

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.

```
virtual IContainerControl&  
    setSingleSelection();
```

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

setTitle

Sets the title for the container that is displayed in all views.

IContainerControl

1	<code>virtual IContainerControl& setTitle(const IResourceId& resourceId);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

2	<code>virtual IContainerControl& setTitle(const char* title);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

setTitleAlignment

Sets the alignment attributes for the title. The default centers the title.

<code>virtual IContainerControl& setTitleAlignment(TitleAlignment alignment = centered);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

showDetailViewTitles

Displays column headings in the 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.

<code>virtual IContainerControl& showDetailViewTitles(Boolean show = true);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support hiding of the column titles.

showTitle

Shows the container title.

<code>virtual IContainerControl& showTitle(Boolean show = true);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

showTitleSeparator

Displays a horizontal separator between the title and the work area.

<code>virtual IContainerControl& showTitleSeparator(Boolean show = true);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

title

Returns the title of the container.

<code>virtual IString title() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

IContainerControl

titleRectangle

Returns the rectangle that contains the container's title.

IRectangle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
titleRectangle() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

Attributes

Use these attribute members to provide valid container styles for IContainerControl::setDefaultAttribute and for the constructor of the IContainerControl (p. 174) class.

defaultAttribute

Returns the default attribute. The default attribute is IContainerControl::classDefaultAttribute (p. 226) unless you have changed it using setDefaultAttribute (p. 187).

static Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultAttribute();	<i>Y</i>	<i>Y</i>	<i>Y</i>

setDefaultAttribute

Sets the default attribute for all subsequent containers.

attribute Use the attributes provided by IContainerControl::Attribute (p. 235) to specify the default attribute.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultAttribute(const Attribute& attribute);	<i>Y</i>	<i>Y</i>	<i>Y</i>

Automatic Deletion Behavior

Use these members to determine behavior that occurs automatically when a container is deleted.

setDeleteColumnsOnClose

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.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDeleteColumnsOnClose(Boolean destroy = true);	<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerControl

setDeleteObjectsOnClose

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.

<code>virtual IContainerControl& setDeleteObjectsOnClose(Boolean destroy = true);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

willDeleteColumnsOnClose

If the columns will be deleted when the container is deleted, true is returned.

<code>Boolean willDeleteColumnsOnClose() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

willDeleteObjectsOnClose

If container objects will be deleted when the container is deleted, true is returned.

<code>Boolean willDeleteObjectsOnClose() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Changing Views

Use these members to change the view.

arrangeIconView

Arranges the icon view. The icons are arranged in horizontal rows from left to right.

<code>virtual IContainerControl& arrangeIconView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>



CM_ARRANGE provides additional information on arrangement in the *Presentation Manager Programming Reference, Volume 3*.

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.

<code>virtual IContainerControl& showDetailsView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerControl

showFlowedNameView

Sets the current view to the flowed name view.

<pre>virtual IContainerControl& showFlowedNameView();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

showFlowedTextView

Sets the current view to the flowed text view.

<pre>virtual IContainerControl& showFlowedTextView();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

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.

<pre>virtual IContainerControl& showIconView();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



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.

showMiniIcons

Shows mini icons in any non-text view. For best results, provide a 16X16 and a 20X20 1-bit-per-pixel version of the image in the .ico file loaded for the `IContainerObject`. When `showMiniIcons` is called, the smaller icons are displayed; otherwise, the regular-sized icon is resized accordingly.

<pre>virtual IContainerControl& showMiniIcons(Boolean mini = true);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>



This function is currently not supported for both the Windows native containers and the `pmCompatible` containers.

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.

IContainerControl

<code>virtual IContainerControl& showNameView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) only support flowed name and flowed text views.

showTextView

Sets the current view to the non-flowed text view.

<code>virtual IContainerControl& showTextView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) only support flowed name and flowed text views.



Text view displays icons.

showTreeIconView

Displays the objects as icons in a tree to represent their relationship to one another.

<code>virtual IContainerControl& showTreeIconView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

showTreeNameView

Sets the current view to the tree name view.

<code>virtual IContainerControl& showTreeNameView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

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.

<code>virtual IContainerControl& showTreeTextView();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>



.*Due to limitations in the Container widget, Tree-text view displays icons.

Comparison

These operators compare two IContainerControls.

IContainerControl

operator == If two containers are the same (that is, their storage location is identical), true is returned.

Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
operator ==(const IContainerControl& container);		Y	Y	Y

Constructors

You can construct and destruct objects of this class.

Note: The AIX version does not support constructing objects from a loaded dialog template or from an existing container control's window handle.

IContainerControl

1	IContainerControl(unsigned long id, IWindow* parent, IWindow* owner, const IRectangle& location = IRectangle (), const Style& style = defaultStyle (), const Attribute& attribute = defaultAttribute ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

Create an IContainerControl on a window by specifying the parent and owner.

2	IContainerControl(unsigned long id, IWindow* parentDialog);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Create an IContainerControl from a container control on a loaded dialog template.

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support this constructor.

3	IContainerControl(const IWindowHandle& handle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	N

Create an IContainerControl from an existing container control's window handle

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support this constructor.

~IContainerControl

IContainerControl

<pre>virtual ~IContainerControl();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Details View Functions

Use these members in the details view.

addColumn 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.

<pre>virtual IContainerControl& addColumn(const IContainerColumn* column, const IContainerColumn* afterColumn = 0);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					



See the IContainerColumn description for more information concerning adding columns to a Windows native container (that is, a container constructed without the pmCompatible style).

columnAt Retrieves the column at the specified 0-based index or cursor.

1	<pre>IContainerColumn* columnAt(const ColumnCursor& cursor) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

2	<pre>IContainerColumn* columnAt(unsigned long index) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

columnCount

Returns the number of columns in the container.

<pre>unsigned long columnCount() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

detailsTitleRectangle

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 the details view when this function is called.

<pre>IRectangle detailsTitleRectangle(Boolean rightSide = false) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

IContainerControl



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar and thus have no right window.

detailsViewSplit

Returns the last column before the split bar.

IContainerColumn*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
detailsViewSplit() const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar.

hideSplitBar Removes the split bar from the details view work area.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hideSplitBar();	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar.

isColumnRight

If the specified column is to the right of the split bar, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isColumnRight(const IContainerColumn* column) const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar and thus have no right window.

removeColumn

Removes the specified column from the container.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
removeColumn(const IContainerColumn* column);	<i>Y</i>	<i>Y</i>	<i>Y</i>

removeColumnAt

Removes the column at a specified cursor position.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
removeColumnAt(IContainerControl::ColumnCursor& cursor);	<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerControl

setDetailsViewSplit

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.

```
virtual IContainerControl&
    setDetailsViewSplit(
        const IContainerColumn* lastColumnBeforeSplit,
        unsigned long pixelsFromLeft = 50 );
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar.

showSplitBar Adds the split bar to the details view work area.

```
virtual IContainerControl&
    showSplitBar( Boolean showSplitBar = true );
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar.

splitBarOffset

Returns the offset of the split bar, in pixels, from the left side of the container window.

```
unsigned long
    splitBarOffset() const;
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar.

Determining the View

Use these members to query the view.

isDetailsView Queries whether the container is currently in the details view.

```
Boolean
    isDetailsView() const;
```

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

isFlowed Queries whether the current container view is flowed. This function applies to the name and text views only.

IContainerControl

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isFlowed() const;	Y	Y	Y

isFlowedNameView

Returns true if the container is currently in the flowed name view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isFlowedNameView() const;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a non-flowed name view.

isFlowedTextView

Returns true if the container is currently in the flowed text view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isFlowedTextView() const;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a non-flowed text view.

isIconView

Queries whether the container is currently in an icon view. If *iconOnly* is false, true is returned for both the icon view and the tree-icon view. If *iconOnly* is true, true is returned only for the icon view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isIconView(Boolean iconOnly = false) const;	Y	Y	Y

isNameView

Queries whether the container is currently in a name view. If *nameOnly* is false, true is returned for both the name view and the tree-name view. If *nameOnly* is true, true is returned for only the name view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isNameView(Boolean nameOnly = false) const;	Y	Y	Y

isShowingMiniIcons

Queries whether the container mode is set for displaying mini icons.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isShowingMiniIcons() const;	Y	Y	I

IContainerControl

isTextView Queries whether the container is currently in a text view. If *textOnly* is false, true is returned for both the text view and the tree-text view. If *textOnly* is true, true is returned for only the text view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTextView(Boolean textOnly = false) const;	Y	Y	Y

isTreeIconView

Returns true only if the container is in the tree-icon view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTreeIconView() const;	Y	Y	Y

isTreeNameView

Returns true only if the container is currently in the tree-name view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTreeNameView() const;	Y	Y	Y

isTreeTextView

Returns true only if the container is currently in the tree-text view.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTreeTextView() const;	Y	Y	Y

isTreeView Queries whether the container is currently in a tree view. This includes the tree-icon, the tree-text, and the tree-name views.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isTreeView() const;	Y	Y	Y

Direct Editing

Use these members in direct editing in the container.

closeEdit Closes an open edit field.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
closeEdit();	Y	Y	Y

IContainerControl

currentEditColumn

Retrieves the column being edited. If a column is not being edited, 0 is returned.

IContainerColumn*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
currentEditColumn() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

currentEditMLE

Retrieves the multiline edit (MLE) field being used for editing. If an edit operation is not in effect, 0 is returned.

IMultiLineEdit*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
currentEditMLE() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

currentEditObject

Retrieves the object being edited. If an object is not being edited, 0 is returned.

IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
currentEditObject() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

editColumnTitle

Opens an edit field on a specified column's heading.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
editColumnTitle(IContainerColumn* column);	<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the column titles.

editContainerTitle

Opens an edit field on the container title.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
editContainerTitle();	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the container title.

editObject

Opens an edit field on a specified object.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
editObject(IContainerObject* object,	<i>Y</i>	<i>Y</i>	<i>Y</i>
IContainerColumn* column = 0);			

IContainerControl

setEditColumn

Stores the specified column where editing occurs.

virtual IContainerControl& setEditColumn(IContainerColumn* column);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

setEditMLE

Stores the specified multiline edit (MLE) field being used for editing.

virtual IContainerControl& setEditMLE(IMultiLineEdit* editField);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------

setEditObject

Stores the specified object where editing occurs.

virtual IContainerControl& setEditObject(IContainerObject* object);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Filtering Objects

Use these members to filter the container.

filter

Filters the container by calling the specified IContainerControl::FilterFn::isMemberOf (p. 243) 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.

1 virtual IContainerControl& filter(const IContainerControl::FilterFn& filterObject);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.

2 virtual IContainerControl& filter();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.

Initialization

Use these members to initialize the container.

initialize

Sets up the container environment. If the first container component you create is an IContainerControl, the library calls this function automatically.

IContainerControl

```
static void  
    initialize();
```

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

Iterating Objects

Use these members to apply behavior to the objects in the container.

allObjectsDo Calls IContainerControl::Iterator::applyTo (p. 245) 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 either of the following occurs:

- IContainerControl::Iterator::applyTo returns true
- All objects in the container have been processed

```
virtual IContainerControl&  
    allObjectsDo( IContainerControl::Iterator& iteratorObject,  
                 Boolean includeDescendents = false );
```

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

Miscellaneous

Use these members to execute miscellaneous IContainerControl operations.

containerFromHandle

Retrieves a container from the list using the specified handle.

```
static IContainerControl*  
    containerFromHandle( const IWindowHandle& handle );
```

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

disableCaching

Stops the dispatch of ICnrQueryDeltaEvents (p. 113) to an ICnrHandler (p. 98).

```
virtual IContainerControl&  
    disableCaching();
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support data caching.

disableDrawBackground

Stops the dispatch of ICnrDrawBackgroundEvent to an ICnrDrawHandler.

```
virtual IContainerControl&  
    disableDrawBackground();
```

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

IContainerControl



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of the container background.

disableDrawItem

Stops the dispatch of ICnrDrawItemEvent to an ICnrDrawHandler.

virtual IContainerControl& disableDrawItem();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects.

enableCaching

If an ICnrHandler (p. 98) has been created and added to the container, this function causes the dispatch of an ICnrQueryDeltaEvent (p. 113) to the handler when the container scrolls past the predefined delta value.

virtual IContainerControl& enableCaching(unsigned long deltaValue = 30);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support data caching.

enableDrawBackground

If an ICnrDrawHandler has been created and added to the container, this function causes the container to dispatch the ICnrDrawBackgroundEvent to its handlers.

virtual IContainerControl& enableDrawBackground(Boolean enable = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of the container background.

enableDrawItem

If an ICnrDrawHandler has been created and added to the container, this function causes the container to dispatch the ICnrDrawItemEvent 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.

virtual IContainerControl& enableDrawItem(Boolean enable = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects.

IContainerControl

isCachingEnabled

Queries whether a delta value is set and caching is enabled.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isCachingEnabled() const;	Y	Y	I



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support data caching.

isDrawBackgroundEnabled

Queries whether owner draw of the container background is enabled.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isDrawBackgroundEnabled() const;	Y	Y	I



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of the container background.

isDrawItemEnabled

Queries whether owner draw of container objects is enabled.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isDrawItemEnabled() const;	Y	Y	I



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects.

numberOfColumnChanges

Retrieves the container's count of column additions and removals.

IContainerControl::ColumnCursor objects use this function to validate their location in the container.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
numberOfColumnChanges() const;	Y	Y	Y

numberOfObjectChanges

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.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
numberOfObjectChanges() const;	Y	Y	Y

IContainerControl

Moving and Copying Subtrees

Use these members to move and copy objects and their descendents.

copyObjectTo

Copies an object and its descendents to a new location in the tree view. The object must override IContainerObject::objectCopy (p. 262). This function returns a copy of the object.

virtual IContainerObject*		<u>Win</u>	<u>PM</u>	<u>Motif</u>
copyObjectTo(IContainerObject* copyObject,		<i>Y</i>	<i>Y</i>	<i>Y</i>
IContainerObject* parentObject = 0,				
IContainerControl* newContainer = 0,				
IContainerObject* afterObject = 0,				
const IPoint& iconViewLocation = IPoint (0 ,				
0));				



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.

isMoveValid

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 (p. 203).

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*. If 0, then *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 source. Otherwise, it is another container.

IContainerControl

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).
- *moveObject* and *afterObject* are the same object.
- 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.

virtual Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
isMoveValid(IContainerObject* moveObject,		Y	Y	Y
IContainerObject* newParentObject = 0,				
IContainerControl* newContainer = 0,				
IContainerObject* afterObject = 0);				

moveObjectTo

Moves the specified object and its descendents to the specified new location. This function calls IContainerControl::isMoveValid (p. 202) 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*. If 0, then *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).

IContainerControl

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.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
moveObjectTo(IContainerObject* moveObject,	<i>Y</i>	<i>Y</i>	<i>Y</i>
IContainerObject* newParentObject = 0,			
IContainerControl* newContainer = 0,			
IContainerObject* afterObject = 0,			
const IPoint& iconViewLocation = IPoint (0 ,			
0));			

Notification Members

These INotificationId strings are used for all notifications that IContainerControl provides to its observers.

enableNotification

Enables the container to send notifications to its observer objects.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
enableNotification(Boolean enable = true);	<i>Y</i>	<i>Y</i>	<i>Y</i>

Object Position

Use these members to manipulate and query an object's position.

columnUnderPoint

Retrieves the column under the specified point, in window coordinates.

IContainerColumn*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
columnUnderPoint(const IPoint& point) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

convertToWorkspace

Converts a rectangle from container window coordinates to workspace coordinates.
This is valid only for the current view.

IContainerControl

```
IRectangle  
convertToWorkspace( const IRectangle& windowRectangle,      Win PM Motif  
                    Boolean rightWindow = false ) const;      Y  Y  Y
```

detailsObjectRectangle

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. It has the following characteristics:

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

This rectangle's width is either of the following:

- 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 the following:

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.

```
1 IRectangle  
    detailsObjectRectangle( const IContainerObject* object,      Win PM Motif  
                          Boolean rightWindow = false ) const;      Y  Y  Y
```



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar and thus have no right window.

IContainerControl

2	<code>IRectangle detailsObjectRectangle(const IContainerObject* object, const IContainerColumn* column) const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	---	-------------------------------	------------------------------	---------------------------------

iconRectangle

Returns the rectangle bounding an object's icon. If you specify *includeText* as true, this rectangle also includes the object's text.

<code>IRectangle iconRectangle(const IContainerObject* object, Boolean includeText = false) const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

moveIconTo Moves the specified object to a new workspace location within the icon view.

<code>virtual IContainerControl& moveIconTo(IContainerObject* object, const IPoint& point);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

objectUnderPoint

Retrieves the object under the specified point, in window coordinates. If the container is in the icon view, this function returns the topmost object.

<code>IContainerObject* objectUnderPoint(const IPoint& point) const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

textRectangle

Returns the text rectangle in window coordinates.

<code>IRectangle textRectangle(const IContainerObject* object) const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

Overrides

This class overrides these inherited members.

backgroundColor

Returns the background color value of the container control. If you have not set this color, the default is returned.

<code>virtual IColor backgroundColor() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>N</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

IContainerControl

foregroundColor

Returns the foreground color value of the container control. If you have not set this color, the default is returned.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
foregroundColor() const;	<i>Y</i>	<i>N</i>	<i>N</i>

isVisible

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.

1	virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	isVisible() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

2	Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	isVisible(const IContainerObject* object) const;	<i>Y</i>	<i>Y</i>	<i>I</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.

resetBackgroundColor

Resets the background color by undoing a previous set.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetBackgroundColor();	<i>Y</i>	<i>N</i>	<i>N</i>

resetForegroundColor

Resets the foreground color by undoing a previous set.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetForegroundColor();	<i>Y</i>	<i>N</i>	<i>N</i>

sendEvent

Overrides IWindow::sendEvent (Vol. II).

1	virtual IEventResult	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	sendEvent(const IEvent& event) const;	<i>N</i>	<i>N</i>	<i>Y</i>

2	virtual IEventResult	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	sendEvent(unsigned long eventId,	<i>N</i>	<i>N</i>	<i>Y</i>
	const IEventParameter1& parameter1 = 0,			
	const IEventParameter2& parameter2 = 0) const;			

IContainerControl

3	<pre>virtual IEventResult sendEvent(EventType eventType, const IEventParameter1& parameter1 = 0, const IEventParameter2& parameter2 = 0) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>N</i></td><td><i>N</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>N</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>N</i>	<i>N</i>	<i>Y</i>						

setBackgroundColor

Sets the background color to the indicated color.

<pre>virtual IContainerControl& setBackgroundColor(const IColor& color);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>Y</i>					



Sets the background color only for PM styled container controls (that is, container controls constructed with the PMCompatible style).

setForegroundColor

Sets the foreground color to the indicated color.

<pre>virtual IContainerControl& setForegroundColor(const IColor& color);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>N</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					



Sets the foreground color only for PM styled container controls (that is, container controls constructed with the PMCompatible style).

Querying Object Attributes

Use these members to query the attributes of an object.

isCollapsed Queries whether the specified tree-view node object is currently collapsed.

<pre>Boolean isCollapsed(const IContainerObject* object) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

isCursored Queries whether the specified object has cursored emphasis.

<pre>Boolean isCursored(const IContainerObject* object) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					





isDropOnAble

Queries whether the specified object is set to receive a direct manipulation drop.

<pre>Boolean isDropOnAble(const IContainerObject* object) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

isExpanded Queries whether the specified tree-view node object is currently expanded.

IContainerControl

	Boolean isExpanded(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
isInUse	Queries whether the specified object has in-use emphasis.			
	Boolean isInUse(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
isSelected	Queries whether the specified object has selection emphasis.			
	Boolean isSelected(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
isSource	Queries whether the specified object has source emphasis. <i>Source emphasis</i> 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.			
	Boolean isSource(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
	This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.			
isTarget	Queries whether the specified object is a target of direct manipulation.			
	Boolean isTarget(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
isVisible	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.			
	Boolean isVisible(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
	The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.			
	virtual Boolean isVisible() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
isWriteable	Returns true if the object can be edited.			

IContainerControl

Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
isWriteable(const IContainerObject* object) const;		<i>Y</i>	<i>Y</i>	<i>Y</i>

Querying the View Port Rectangle

Use these members to determine the work area relative to either the work space or the window.

detailsViewPortOnWindow

Retrieves either the right or left work area of a split details view, in window coordinates.

IRectangle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
detailsViewPortOnWindow(Boolean rightSide = false) const;		<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar and thus have no right window.

detailsViewPortOnWorkspace

Retrieves either the right or left work area of a split details view, in workspace coordinates.

IRectangle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
detailsViewPortOnWorkspace(Boolean rightSide = false) const;		<i>Y</i>	<i>Y</i>	<i>Y</i>



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a split bar and thus have no right window.

viewPortOnWindow

Retrieves the current work area in window coordinates.

IRectangle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
viewPortOnWindow() const;		<i>Y</i>	<i>Y</i>	<i>Y</i>

viewPortOnWorkspace

Retrieves the current work area in workspace coordinates.

IRectangle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
viewPortOnWorkspace() const;		<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerControl

Refresh

Use these members to refresh the container.

isRefreshOn Queries the refresh state of the container. If the container immediately displays visual changes, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isRefreshOn() const;	Y	Y	Y

refresh Refreshes the container by forcing a repaint.

1	virtual IContainerControl& refresh(IContainerObject* object, Boolean immediate = false);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

Refreshes an object in the container. If *immediate* is set to true, the painting is done synchronously; that is, all container painting is completed before this function returns to the caller. If *immediate* is false, the container painting occurs asynchronously, and painting may not be completed when the function returns to the caller.

2	virtual IContainerControl& refresh(Boolean immediate = false);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

Refreshes all objects in the container. If *immediate* is set to true, the painting is done synchronously; that is, all container painting is completed before this function returns to the caller. If *immediate* is false, the container painting occurs asynchronously, and painting may not be completed when the function returns to the caller.

3	virtual IContainerControl& refresh(const IRectangle& invalidRectangle, Boolean immediate);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	I

Refreshes a specific rectangle in a container. If *immediate* is set to true, the painting is done synchronously; that is, all container painting is completed before this function returns to the caller. If *immediate* is false, the container painting occurs asynchronously, and painting may not be completed when the function returns to the caller.

4	virtual IContainerControl& refresh(IWindow::RefreshType type);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	I

Refreshes the window based on the IWindow::RefreshType (Vol. II).

IContainerControl

refreshAllContainers

Refreshes all containers.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
refreshAllContainers(Boolean immediate = false);	Y	Y	Y

Refreshes all objects in all containers. If *immediate* is set to true, the painting is done synchronously; that is, all container painting is completed before this function returns to the caller. If *immediate* is false, the container painting occurs asynchronously, and painting may not be completed when the function returns to the caller.

setRefreshOff

Disables refreshing the container until IContainerControl::refresh (p. 211) or setRefreshOn (p. 212) 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.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setRefreshOff();	Y	Y	Y

setRefreshOn

Enables refreshing the container after changes. This function sets the container's refresh state to on. Call IContainerControl::refresh after this function to show changes made to the container while refresh was set to off.

virtual IContainerControl&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setRefreshOn(Boolean on = true);	Y	Y	Y

Retrieving Objects

Use these members to retrieve objects in the container.

containsObject

If the specified object is in the container, true is returned.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
containsObject(const IContainerObject* object) const;	Y	Y	Y

cursoredObject

Returns the object on which the cursor is located.

IContainerControl

virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
cursoredObject() const;	<i>Y</i>	<i>Y</i>	<i>I</i>

descendentsOf

Returns the set of all descendents of an object in the tree view.

ICnrObjectSet	<u>Win</u>	<u>PM</u>	<u>Motif</u>
descendentsOf(IContainerObject* parentObject) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

immediateDescendentsOf

Returns the set of immediate descendents *only* of the specified object in the tree view.

ICnrObjectSet	<u>Win</u>	<u>PM</u>	<u>Motif</u>
immediateDescendentsOf(IContainerObject* parentObject) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

objectAt

Returns an object at the specified index or cursor position in the container.

1	virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	objectAt(unsigned long index) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

2	virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	objectAt(const IContainerControl::ObjectCursor& cursor) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

3	virtual IContainerObject*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	objectAt(const IContainerControl::TextCursor& cursor) const;	<i>Y</i>	<i>Y</i>	<i>I</i>

objectCount

Returns the number of objects in the container.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
objectCount() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

objectList

Returns the set of all objects in the container.

ICnrObjectSet	<u>Win</u>	<u>PM</u>	<u>Motif</u>
objectList() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

parentObject

Returns the parent of an object in the tree view. If the object does not have a parent object, true is returned.

IContainerControl

```
virtual IContainerObject*
    parentObject( const IContainerObject* childObject ) const;      Win PM Motif
                                                                    Y   Y   Y
```

Scrolling

Use these members to scroll the container.

scroll

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 the following:

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.

```
virtual IContainerControl&
    scroll( long verticalPixels,
           long horizontalPixels,
           Boolean rightSide = false );      Win PM Motif
                                                                    Y   Y   Y
```

scrollDetailsHorizontally

Scrolls either the right or left side of a split details view horizontally.

The parameters are the following:

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.

```
virtual IContainerControl&
    scrollDetailsHorizontally( long horizontalPixels,
                              Boolean rightSide = false );      Win PM Motif
                                                                    Y   Y   Y
```

IContainerControl

scrollHorizontally

Scrolls the container horizontally.

The parameters are the following:

pixels The number of pixels to scroll horizontally.

rightSide If this container is in the details view, specify true to scroll the right pane and false to scroll the left pane.

<pre>virtual IContainerControl& scrollHorizontally(long pixels, Boolean rightSide = false);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

scrollToObject

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.

1	<pre>virtual IContainerControl& scrollToObject(const IContainerObject* object, const IContainerColumn* column, Boolean leftJustify = true);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						

2	<pre>virtual IContainerControl& scrollToObject(const IContainerObject* object);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						

scrollVertically

Scrolls the container vertically.

<pre>virtual IContainerControl& scrollVertically(long pixels);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

Setting Object Attributes

Use these members to set and remove the attributes of objects.

collapse

Collapses the descendents of an object in the tree view.

To modify the icons used for expanding and collapsing the tree view, see `IContainerControl::setTreeItemIcons` (p. 221).

Note: The parent is not actually being collapsed or expanded.

IContainerControl

<code>virtual IContainerControl& collapse(IContainerObject* object);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

disableDataUpdate

Disables direct editing of the text of an object.

<code>virtual IContainerControl& disableDataUpdate(IContainerObject* object);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

disableDrop Disables an object from receiving a drop. No target emphasis is drawn.

<code>virtual IContainerControl& disableDrop(IContainerObject* object);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

enableDataUpdate

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`.

<code>virtual IContainerControl& enableDataUpdate(IContainerObject* object, Boolean enable = true);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

enableDrop Enables a specified object to be the target of a drag event.

<code>virtual IContainerControl& enableDrop(IContainerObject* object, Boolean enable = true);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	-------------------------------	------------------------------	---------------------------------

expand Expands the tree view to show the descendents of the specified parent object.

To modify the icons used for expanding and collapsing the tree view, see `IContainerControl::setTreeItemIcons` (p. 221).

<code>virtual IContainerControl& expand(IContainerObject* object);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

hideObject Makes a specified object invisible by filtering it out of the container.

<code>virtual IContainerControl& hideObject(IContainerObject* object);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

IContainerControl

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.

hideSourceEmphasis

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.

1 virtual IContainerControl&
hideSourceEmphasis(); Win PM Motif
Y Y I

Win This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

2 virtual IContainerControl&
hideSourceEmphasis(IContainerObject* object); Win PM Motif
Y Y I

Win This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

removeInUse Removes in-use emphasis from the specified object.

virtual IContainerControl&
removeInUse(IContainerObject* object); Win PM Motif
Y Y I

removeSelected

Removes selection emphasis from the specified object.

virtual IContainerControl&
removeSelected(IContainerObject* object); Win PM Motif
Y Y Y

PM A single-selection container or a container in the tree view must always have exactly one object selected. You cannot remove selection emphasis from the only selected object in these cases.

Win A single-selection container or a container in tree view must always have exactly one object selected. You cannot remove selection emphasis from the only selected object in these cases.

setCursor Gives the specified object cursor emphasis.

virtual IContainerControl&
setCursor(IContainerObject* object); Win PM Motif
Y Y I

IContainerControl

setInUse Gives the specified object in-use emphasis.

<pre>virtual IContainerControl& setInUse(IContainerObject* object, Boolean inUse = true);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) do support the in-use emphasis but require that the icon have transparency. The hashed background is only displayed where the object's icon has transparency.

setSelected Gives the specified object selection emphasis.

<pre>virtual IContainerControl& setSelected(IContainerObject* object, Boolean select = true);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					



A single selection container or a container in the tree view must always have exactly one object selected. You cannot remove selection emphasis from the only selected object in these cases.



A single-selection container or a container in tree view must always have exactly one object selected. You cannot remove selection emphasis from the only selected object in these cases.

showObject Shows the specified object that is currently invisible or filtered from the container.

<pre>virtual IContainerControl& showObject(IContainerObject* object, Boolean visible = true);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of container objects.

showSourceEmphasis

Draws source emphasis. If you specify an object, the emphasis is drawn around the object. Otherwise, the container itself gets the source emphasis.



<pre>virtual IContainerControl& showSourceEmphasis(Boolean source = true);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

IContainerControl

2	<code>virtual IContainerControl& showSourceEmphasis(IContainerObject* object, Boolean source = true);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>I</i>						

Win This function is ignored for both the Windows native containers and the pmCompatible containers. Source emphasis is not supported in either case.

Sorting Objects

Use these members to provide sorting in the container.

nlsCompare Compares two specified strings consisting of a national character set. Use this function for implementing National Language Support (NLS).

<code>static long nlsCompare(const char* text1, const char* text2);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

sort Sorts the container using a user-defined sort function, IContainerControl::CompareFn (p. 240).

<code>virtual IContainerControl& sort(const IContainerControl::CompareFn& sortObject);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

sortByIconText

Sorts the container by the icon's text.

<code>virtual IContainerControl& sortByIconText(Boolean ascending = true);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Styles

These style members provide valid container styles for IContainerControl::setDefaultStyle and for the constructor of the IContainerControl (p. 174) class.

convertToGUIStyle

Converts style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles, those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

<code>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

IContainerControl

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 231) unless you have changed it using setDefaultStyle (p. 220).

<pre>static Style defaultStyle();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setDefaultStyle

Sets the default style for all subsequent containers.

style Use the styles provided by IContainerControl::Style (p. 250) to specify the default style.

<pre>static void setDefaultStyle(const Style& style);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Tree View Functions

Use these members in the tree view.

collapseTree Collapses all nodes of a tree view.

<pre>virtual IContainerControl& collapseTree();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

expandTree Expands all nodes of a tree view.

<pre>virtual IContainerControl& expandTree();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

hideTreeLine Does not show the lines connecting parents to children's records.

<pre>virtual IContainerControl& hideTreeLine();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setTreeExpandIconSize

Changes the size of the expanded and collapsed icons.

<pre>virtual IContainerControl& setTreeExpandIconSize(const ISize& sizeIcon);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support sizing the tree view's expand and collapse icons.

IContainerControl

setTreeItemIcons

Sets the expanded and collapsed icons for a tree view.

1	<code>virtual IContainerControl& setTreeItemIcons(const IPointerHandle& expanded, const IPointerHandle& collapsed);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support changing the tree view expand and collapse icons.

2	<code>virtual IContainerControl& setTreeItemIcons(const IResourceId& expanded, const IResourceId& collapsed);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support changing the tree view's expand and collapse icons.

setTreeViewIndent

Sets the distance that children are offset horizontally from their parent.

<code>virtual IContainerControl& setTreeViewIndent(long indentPixels = - 1);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

Win The native Windows containers (that is, containers constructed without the pmCompatible style) do not support changing the indentation used for tree view child objects.

showTreeLine

Shows the lines connecting parents to children's records.

<code>virtual IContainerControl& showTreeLine(long treeLinePixelWidth = - 1);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	-------------------------------	------------------------------	---------------------------------

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

IContainerControl

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Compound Control Method

These members return miscellaneous informaton for this compound control.

topHandle 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.

virtual IWindowHandle
topHandle() const;

Win PM Motif
N N Y

Constructors

You can construct and destruct objects of this class.

Note: The AIX version does not support constructing objects from a loaded dialog template or from an existing container control's window handle.

:h14IContainerControl

IContainerControl();

Win PM Motif
Y Y Y

Provides the default constructor, which accepts no arguments.

IContainerControl

Drag and Drop Support

Use these members to customize the conditions for detecting a drag operation.

isDragStarting

Queries whether conditions are met to satisfy a drag operation.

A drag operation will occur if the mouse pointer is over a container object when the button is pressed.

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

Event-Handling Implementation

Event-handling implementation members perform processing that allows handlers to receive GUI events and to route these events.

registerCallbacks

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

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

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
registerCallbacks();	<i>N</i>	<i>N</i>	<i>Y</i>

unregisterCallbacks

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

void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
unregisterCallbacks();	<i>N</i>	<i>N</i>	<i>Y</i>

Implementation

These members implement the class.

attributes Returns an attribute mask for the specified object.

IContainerControl

unsigned long attributes(const IContainerObject* object) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

baseRecordSize

Returns the size of the RECORDCORE portion of the objects stored in a container.

virtual unsigned long baseRecordSize();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

calcMinimumSize

Returns a minimum size for the container.

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

column

Returns a column matching the command criteria.

IContainerColumn* column(IContainerColumn* column, unsigned long command, Boolean visible);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

containerAttributes

Returns the current container attribute mask.

unsigned long containerAttributes() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

containerKey Returns the key used to control access when updating the container. The key acts as a semaphore to the container updates.

static IPrivateResource& containerKey();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

containerList Returns the list of containers in this application.

static ICnrControlList& containerList();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

IContainerControl

hwndAllocation

Returns the container used for object allocation.

<pre>static IWindowHandle hwndAllocation();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setAttributes Updates the attributes of the specified object.

<pre>virtual void setAttributes(IContainerObject* object, unsigned long attributeToTurnOn, unsigned long attributeToTurnOff);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setContainerAttributes

Sets the attributes into the container control.

<pre>virtual void setContainerAttributes(unsigned long attributeToTurnOff, unsigned long attributeToTurnOn);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setEmphasis Sets the specified object to the specified emphasis attribute.

<pre>virtual void setEmphasis(IContainerObject* object, unsigned long emphasisAttribute, Boolean setOn = true);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Attributes

Use these attribute members to provide valid container styles for IContainerControl::setDefaultAttribute and for the constructor of the IContainerControl (p. 174) class.

IContainerControl

alignTitleCentered

Centers the container title.

<code>static const Attribute alignTitleCentered;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	------------------------	-----------------------	--------------------------

alignTitleLeft Left-justifies the container title.

<code>static const Attribute alignTitleLeft;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	------------------------	-----------------------	--------------------------

alignTitleRight

Right-justifies the container title.

<code>static const Attribute alignTitleRight;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

classDefaultAttribute

Specifies the original default attribute for this class, which is orderedTargetEmphasis | detailsViewTitles | visibleTreeLine | readOnlyTitle | iconView.

<code>static const Attribute classDefaultAttribute;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

detailsView Shows the details view of the container.

<code>static const Attribute detailsView;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

detailsViewTitles

Shows the details view titles.


<code>static const Attribute detailsViewTitles;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
--	------------------------	-----------------------	--------------------------

flowedView Shows the flowed view of the container.

Note: This attribute is only valid when combined with the nameView attribute or the textView attribute.

IContainerControl


static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
flowedView;	Y	Y	Y


 The native Windows containers (that is, containers constructed without the pmCompatible style) only support flowed name and flowed text views.

handleDrawBackground

Causes draw-background messages to be dispatched to objects of classes derived from ICnrDrawHandler.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleDrawBackground;	Y	Y	Y


 The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of the container background.


 The AIX release does not support this attribute because AIX does not support the ICnrDrawHandler.

handleDrawItem

Causes draw-item messages to be dispatched to objects of classes derived from ICnrDrawHandler.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleDrawItem;	Y	Y	Y

 The native Windows containers (that is, containers constructed without the pmCompatible style) do not support owner drawing of container objects.

 The AIX release does not support this attribute because AIX does not support the ICnrDrawHandler.

iconView Shows the icon view of the container.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
iconView;	Y	Y	Y

miniIcons Shows mini icons.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
miniIcons;	Y	Y	Y

IContainerControl

mixedTargetEmphasis

Sets the mixed-target emphasis.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
mixedTargetEmphasis;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support mixed-target emphasis for drag and drop.

nameView

Shows the name view of the container.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
nameView;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) only support flowed name and flowed text views.

orderedTargetEmphasis

Sets the ordered-target emphasis.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
orderedTargetEmphasis;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support ordered-target emphasis for drag and drop.

readOnlyTitle Disables the editing of the container title.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
readOnlyTitle;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support editing of the container title.

textView

Shows the text view of the container.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
textView;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) only support flowed name and flowed text views.

IContainerControl

titleSeparator

Adds a separator line after the title.

```
static const Attribute  
    titleSeparator;
```

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

treeView

Shows the tree view of the container.

```
static const Attribute  
    treeView;
```

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

visibleTitle

Shows the container title.

```
static const Attribute  
    visibleTitle;
```

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

visibleTreeLine

Shows the tree line in the tree view.

```
static const Attribute  
    visibleTreeLine;
```

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

Notification Members

These INotificationId strings are used for all notifications that IContainerControl provides to its observers.

addId

Notification identifier provided to observers when objects are added to a container. IContainerControl provides an unsigned long value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value indicates how many objects were added.

```
static INotificationId const  
    addId;
```

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

detailsViewTitlesId

Notification identifier provided to observers when the visibility of the titles of the container details view change. IContainerControl provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the details view titles are now visible and false if the details view titles are invisible.

IContainerControl

	<pre>static INotificationId const detailsViewTitlesId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
enterId	Notification identifier provided to observers when an item in the container is double clicked, or the user presses the Enter key. IContainerControl provides the enter event in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).			
	<pre>static INotificationId const enterId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
removeId	Notification identifier provided to observers when objects are removed from a container. IContainerControl provides an unsigned long value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value indicates how many objects were removed.			
	<pre>static INotificationId const removeId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
selectId	Notification identifier provided to observers when a container item's selection state changes. IContainerControl provides the emphasis event in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).			
	<pre>static INotificationId const selectId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
titleId	Notification identifier provided to observers when the text of the title changes in the container control. IContainerControl provides a pointer to the new title's text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).			
	<pre>static INotificationId const titleId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
titleVisibleId	Notification identifier provided to observers when the visibility of the container's title changes. IContainerControl provides a Boolean value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the title is now visible and false if the title is invisible.			
	<pre>static INotificationId const titleVisibleId;</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

Styles

These style members provide valid container styles for IContainerControl::setDefaultStyle and for the constructor of the IContainerControl (p. 174) class.

IContainerControl

autoPosition Automatically positions objects added to the container in the icon view.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
autoPosition;	Y	Y	Y

classDefaultStyle

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

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

extendedSelection

Specifies that the container use extended selection. *Extended selection* is an enhanced version of single selection that lets the user select discontinuous sets of container items.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
extendedSelection;	Y	Y	Y

multipleSelection

Specifies that the container use multiple selection. *Multiple selection* is a mode that allows the user unrestricted selection of objects.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
multipleSelection;	Y	Y	Y



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support multiple selection. Extended selection is used instead.

noSharedObjects

Specifies that this container's objects are not shared with other containers. This provides better performance when the container is deleted or deleteAllObjects is called.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noSharedObjects;	Y	Y	Y

IContainerControl

pmCompatible

Provides a PM-like container control. Do not use this style for native Windows containers.

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

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

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

singleSelection

Specifies that 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 always selected. If you remove an object with selection emphasis, the container selects another object.

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

verifyPointers

Validates that container objects exist. If you specify this style, the presentation system checks that an object exists in the collection before using it. If a container record does not exist, an `IAccessError` exception is thrown.

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



The native Windows containers (that is, containers constructed without the `pmCompatible` style) do not support a `verify pointers` style.

Inherited Public Data

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IContainerControl contains the following nested classes:

- IContainerControl::FilterFn (see page 242)
- IContainerControl::ObjectCursor (see page 246)
- IContainerControl::Style (see page 250)
- IContainerControl::ColumnCursor (see page 236)
- IContainerControl::Iterator (see page 244)
- IContainerControl::Attribute (see page 235)
- IContainerControl::TextCursor (see page 251)
- IContainerControl::CompareFn (see page 240)

EnumerationOrder

IContainerControl

```
EnumerationOrder {  
    itemOrder,  
    zOrder  
};
```

Use these enumerators to specify the order in which icons are enumerated:

itemOrder

Enumerates 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

Enumerates container records by Z-order, first to last.

Z-order 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.



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support a zOrder enumeration.

TitleAlignment

```
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.



IContainerControl::Attribute

Derivation IBase
 IBitFlag
 IContainerControl::Attribute

Inherited By None.

Header File icnrcctl.hpp

The nested class IContainerControl::Attribute provides a set of valid attributes for the IContainerControl (p. 174) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IContainerControl::ColumnCursor

IContainerControl::ColumnCursor

Derivation

IBase
IVBase
IContainerControl::ColumnCursor

Inherited By None.

Header File icnrcctl.hpp

Members

Member	Page	Member	Page
Constructor	237	previous	237
ColumnCursor	237	setCurrent	237
current	236	setToFirst	238
first	237	setToLast	238
invalidate	239	setToNext	238
isValid	239	setToPrevious	238
last	237	~ColumnCursor	238
next	237		

The IContainerControl::ColumnCursor class navigates through the container's details view columns.



Iterating of columns in a native Windows container (that is, a container constructed without the pmCompatible style) may not be as expected. Any icon column added other than the valid one is considered invalid and not visible and will be iterated after the valid columns are iterated. The native Windows container also has a required first column. If IContainerColumn(s) are not inserted for these, then the cursor will not iterate them. See IContainerColumn documentation for more details.



The leftmost column that the Container widget maintains automatically is never accessed by the ColumnCursor.

Public Functions

Column Retrieval

Use these members to move the cursor and retrieve columns.

current Returns the current column of this cursor.

IContainerControl::ColumnCursor

	<pre>virtual IContainerColumn* current() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						
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.							
	<pre>virtual IContainerColumn* first();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						
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.							
	<pre>virtual IContainerColumn* last();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						
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.							
	<pre>virtual IContainerColumn* next();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						
previous	Points to and returns the previous or last column in the container.							
	<pre>virtual IContainerColumn* previous();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						
setCurrent	Sets the cursor to point to the specified column and validates the cursor. This function does not check for visibility.							
	<pre>virtual ColumnCursor& setCurrent(const IContainerColumn* currentColumn);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						

Constructors

You can construct and destruct objects of this class.

ColumnCursor

Constructs 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

IContainerControl::ColumnCursor

- Optionally, a Boolean indicating whether all columns, or only visible columns, are navigated

ColumnCursor(const IContainerControl& container, Boolean visibleOnly = false);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

~ColumnCursor

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

Cursor Movement

Use these members to move the cursor and retrieve columns.

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.

virtual Boolean setToFirst();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------------------------------	-----------------	----------------	-------------------

setToLast Points to the last column in the container. If there is a last column, true is returned. Otherwise, false is returned.

virtual Boolean setToLast();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---------------------------------	-----------------	----------------	-------------------

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.

virtual Boolean setToNext();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---------------------------------	-----------------	----------------	-------------------

setToPrevious

Points to the previous or last column in the container.

virtual Boolean setToPrevious();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
-------------------------------------	-----------------	----------------	-------------------

IContainerControl::ColumnCursor

Cursor Validation

Use these members to validate the cursor.

invalidate Flags the cursor as not valid.

```
virtual ColumnCursor&
    invalidate();
```

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

isValid 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.

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

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

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::CompareFn

IContainerControl::CompareFn

Derivation IBase
IVBase
IContainerControl::CompareFn

Inherited By None.

Header File icnrcctl.hpp

Members	Member	Page	Member	Page
	Constructor	240	isEqual	241
	CompareFn	240	~CompareFn	240

The IContainerControl::CompareFn class provides 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 precedes the other.

When the container calls IContainerControl::sort (p. 219), an IContainerControl::CompareFn object is passed as input to this function. Thereafter, the container calls IContainerControl::CompareFn::isEqual (p. 241), as needed, to sort each container object.

Public Functions

Constructors

You can construct and destruct objects of this class.

CompareFn Provides the default constructor.

CompareFn();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~CompareFn

virtual ~CompareFn();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

IContainerControl::CompareFn

Object Sorting

Use these members to compare objects in the container.

isEqual 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.

```
virtual int  
    isEqual( IContainerObject* object1,  
            IContainerObject* object2,  
            IContainerControl* container ) const = 0;
```

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

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::FilterFn

IContainerControl::FilterFn

Derivation IBase
IVBase
IContainerControl::FilterFn

Inherited By None.

Header File icnrcctl.hpp

Members		Member	Page	Member	Page
		Constructor	242	isMemberOf	243
		FilterFn	242	~FilterFn	243

The IContainerControl::FilterFn class provides 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 (p. 198), an IContainerControl::FilterFn object is passed as input to this function. Thereafter, the container calls IContainerControl::FilterFn_isMemberOf (p. 243) as needed to filter each container object.



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of objects.

Public Functions

Constructors

You can construct and destruct objects of this class.

FilterFn Provides the default constructor.

FilterFn();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

IContainerControl::FilterFn

~FilterFn

```
virtual  
~FilterFn();
```

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

Object Filtering

Use these members to filter objects from the container.

isMemberOf If the specified object is to remain in the container, true is returned. If the object is to be removed, false is returned.

```
virtual Boolean  
isMemberOf( IContainerObject* object,  
            IContainerControl* container ) const = 0;
```

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

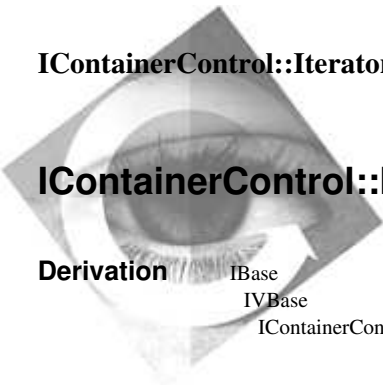
Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::Iterator

IContainerControl::Iterator

Derivation IBase
IVBase
IContainerControl::Iterator

Inherited By None.

Header File icnrcctl.hpp

Members	Member	Page	Member	Page
	Constructor	244	Iterator	244
	applyTo	245	~Iterator	244

The IContainerControl::Iterator class applies 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 (p. 245). Next, call IContainerControl::allObjectsDo (p. 199) 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.

Public Functions

Constructors

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

Iterator You can not construct objects of this class because it is an abstract base class.

Iterator();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~Iterator

virtual ~Iterator();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

IContainerControl::Iterator

Filter Behavior

Use these members apply a behavior to all objects in a container.

applyTo If the iteration is to continue, true is returned. If the iteration is to stop, false is returned.

```
virtual Boolean  
    applyTo( IContainerObject* object ) = 0;
```

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

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::ObjectCursor

IContainerControl::ObjectCursor

Derivation IBase
IVBase
IContainerControl::ObjectCursor

Inherited By None.

Header File icnrcctl.hpp

Members	Member	Page	Member	Page
	Constructor	246	previous	249
	current	248	setCurrent	249
	first	248	setToFirst	247
	invalidate	248	setToLast	247
	isValid	248	setToNext	247
	last	248	setToPrevious	247
	next	249	~ObjectCursor	247
	ObjectCursor	246		

The IContainerControl::ObjectCursor class iterates 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

ObjectCursor Construct an object of this class to iterate objects in the container.

```
ObjectCursor(  
    const IContainerControl& container,  
    IContainerObject::Emphasis emphasis =  
        IContainerObject::none,  
    EnumerationOrder enumeration = itemOrder );
```

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

Construct an object of this class to iterate all records either in the container or by meeting the emphasis selection criteria. Optionally, you can specify whether to iterate records in item order (the default) or Z-order.

IContainerControl::ObjectCursor

2	<code>ObjectCursor(const IContainerControl& container, const IContainerObject* parentObject, Boolean allDescendents = false);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Construct an object of this class to iterate all objects having the specified object as a parent. Optionally, you can specify whether to iterate just the immediate child objects or all descendents of the specified parent.

~ObjectCursor

<code>virtual ~ObjectCursor();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Cursor Movement

Use these members to control cursor movement based on the type of objects indicated in the construction of the cursor.

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.

<code>virtual Boolean setToFirst();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setToLast Points to the last object in the container. If there is a last object, true is returned. Otherwise, false is returned.

<code>virtual Boolean setToLast();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

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.

<code>virtual Boolean setToNext();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setToPrevious

Points to the previous or last object in the container.

<code>virtual Boolean setToPrevious();</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

IContainerControl::ObjectCursor

Cursor Validation

Use these members to validate the cursor.

invalidate Flags the cursor as not valid.

<pre>virtual ObjectCursor& invalidate();</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

isValid 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.

<pre>virtual Boolean isValid() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Object Retrieval

Use these members to retrieve objects.

current Returns the current object of this cursor.

<pre>virtual IContainerObject* current() const;</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

first 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.

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

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.

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

IContainerControl::ObjectCursor

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.

<code>virtual IContainerObject* next();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

previous Points to and returns the previous or last object in the container.

<code>virtual IContainerObject* previous();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setCurrent Sets the cursor to point to the specified object and validates the cursor. This function does not check for visibility.

<code>virtual ObjectCursor& setCurrent(const IContainerObject* currentObject);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::Style

IContainerControl::Style

Derivation IBase
IBitFlag
IContainerControl::Style

Inherited By None.

Header File icnrctl.hpp

The nested class IContainerControl::Style provides a set of valid styles for the IContainerControl (p. 174) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

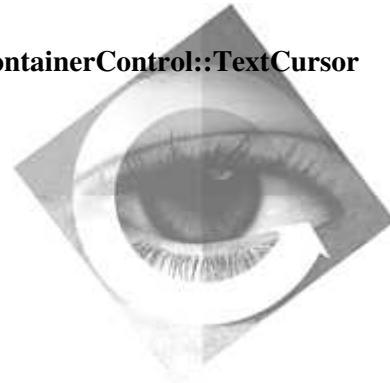
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControl::TextCursor

Derivation IBase
IVBase
IContainerControl::TextCursor

Inherited By None.

Header File icnrcctl.hpp

Members	Member	Page	Member	Page
	Constructor	251	setCurrent	254
	current	253	setToFirst	252
	first	253	setToLast	252
	invalidate	253	setToNext	252
	isValid	253	setToPrevious	253
	last	254	TextCursor	251
	next	254	~TextCursor	252
	previous	254		

The IContainerControl::TextCursor class iterates through the container and finds objects containing a specified text string. You specify this text string when creating the text cursor.

Public Functions

Constructors

You can construct and destruct objects of this class.

TextCursor Constructs objects of this class by specifying a text string to search the container for. Optionally, you can specify that the search return any of the following:

- 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

IContainerControl::TextCursor

- Create a new TextCursor

If you create a details view TextCursor, it searches all the text columns for the specified string.

TextCursor(const IContainerControl& container, const char* text, Boolean isCaseSensitive = true, Boolean isFirstInRecord = false, Boolean isExactMatch = false);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

~TextCursor

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

Cursor Movement

Use these members to position the text cursor.

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.

virtual Boolean setToFirst();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------------------------------	-----------------	----------------	-------------------

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.

virtual Boolean setToLast();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---------------------------------	-----------------	----------------	-------------------

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.

virtual Boolean setToNext();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---------------------------------	-----------------	----------------	-------------------

IContainerControl::TextCursor

setToPrevious

Points to the previous or last object in the container.

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

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

Cursor Validation

Use these members to validate the cursor.

invalidate

Flags the cursor as not valid.

```
virtual TextCursor&  
    invalidate();
```

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

isValid

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.

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

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



The AIX release does not support this attribute because AIX does not support the ICnrDrawHandler.

Object Retrieval

Use these members to retrieve objects using a text cursor.

current

Returns the current object of this cursor.

```
virtual IContainerObject*  
    current() const;
```

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

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.

```
virtual IContainerObject*  
    first();
```

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

IContainerControl::TextCursor

last Points to and returns the last object in the container with matching text. Returns 0 if there is no object in the container. This function validates a cursor that was set invalid as a result of adding or removing an object from the container.

```
virtual IContainerObject*
    last();
```

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

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.

```
virtual IContainerObject*
    next();
```

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

previous Points to and returns the previous or last object in the container with the matching text.

```
virtual IContainerObject*
    previous();
```

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

setCurrent Sets the cursor to point to the specified object.

```
virtual TextCursor&
    setCurrent( const IContainerObject* currentObject );
```

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

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IContainerControlNotifyHandler

Derivation

- IBase
- IVBase
- IHandler
- IWindowNotifyHandler
- IContainerControlNotifyHandler

Inherited By None.

Header File icnrnhdr.hpp

Members	Member	Page
	Constructor	255
	dispatchHandlerEvent	256
	~IContainerControlNotifyHandler	255

The IContainerControlNotifyHandler class processes events for all classes of containers.

This class is designed to handle events that require the container class to generate a notification. If notifications are enabled for this class, a notification is generated and sent to all observers when the proper conditions for the specific notification exist.

Public Functions

Constructors

You can construct and destruct objects of this class.

IContainerControlNotifyHandler

Provides the default constructor.

```
IContainerControlNotifyHandler();
```

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

~IContainerControlNotifyHandler

IContainerControlNotifyHandler

```
virtual  
    ~IContainerControlNotifyHandler();
```

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

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

This function evaluates the event to determine if it is appropriate for this handler object to process.

dispatchHandlerEvent

If any of the following events are received, the container control observers are notified:

- remove object event
- add object event
- select event
- enter event
- title visible event
- details view title event
- title event

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

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

IContainerControlNotifyHandler

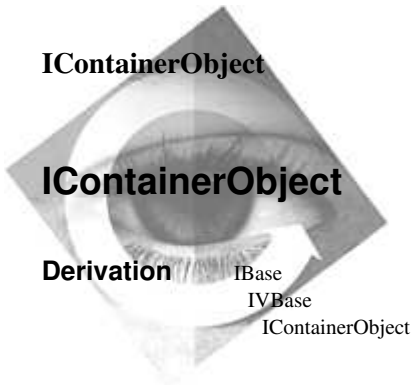
Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File icnrobj.hpp

Members				
Member	Page	Member	Page	
Constructor	259	isRefreshOn	265	
baseRecord	267	isVisible	265	
decrementUseCount	268	isWriteable	265	
disableDataUpdate	263	objectCopy	262	
disableDrop	263	operator ==	262	
enableDataUpdate	263	operator delete	262	
enableDrop	263	operator new	262	
handleCursoredChange	260	refresh	265	
handleInuseChange	261	removeInUse	265	
handleOpen	261	setAttributes	268	
handleSelectedChange	261	setBase	268	
handleTreeCollapse	261	setClosed	265	
handleTreeExpand	261	setEmphasis	268	
helpId	262	setIcon	266	
hide	264	setIconText	266	
icon	264	setInUse	266	
iconText	264	setOpen	266	
incrementUseCount	268	setRefreshOff	267	
initialize	268	setRefreshOn	267	
isAttribute	268	show	267	
isDropOnAble	264	useCount	269	
isInUse	264	~IContainerObject	260	
isOpen	264			

The IContainerObject class 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 (p. 153)

IContainerObject

Note: You can only edit the string type. We recommend that you use the class IString (Vol. I) 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:

- Satisfy 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 (p. 174) 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.

Public Functions

Constructors

You can construct and destruct objects of this class.

IContainerObject

1	IContainerObject();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

This provides the default constructor, which accepts no parameters.

2	IContainerObject(const IString& string, const IPointerHandle& iconHandle = 0);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

Construct an IContainerObject object by providing an IString (Vol. I) for text and an IPointerHandle (Vol. II) for the icon.

3	IContainerObject(const IString& string, const IResourceId& iconId);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerObject

Construct an IContainerObject object by providing an IString for text and an IResourceId (Vol. II) for the icon.

4	IContainerObject(const IResourceId& nameID, const IResourceId& iconId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	--	-----------------	----------------	-------------------

Construct an IContainerObject object by providing resource IDs for both the text and the icon.

5	IContainerObject(const IContainerObject& object);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	---	-----------------	----------------	-------------------

Construct an IContainerObject object by copying an existing IContainerObject or a class derived from IContainerObject. If you derive from IContainerObject, provide a copy constructor in the class to ensure that the object is copied correctly.

6	IContainerObject(const IString& string, unsigned long iconId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	---	-----------------	----------------	-------------------

Construct an IContainerObject object by 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 because both IResourceIds and IPointerHandles can be built implicitly from an unsigned long integer.

~IContainerObject

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

Derived Class Provided Members

These members must be provided by a derived class if the behavior is required.

handleCursoredChange

Called when cursored emphasis changes on an object and the IContainerControl (p. 174) containing the object has an ICnrHandler (p. 98). If the object has just acquired cursored emphasis, *acquired* is set to true.

virtual void handleCursoredChange(IContainerControl* container, Boolean acquired);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
--	-----------------	----------------	-------------------

IContainerObject

handleInuseChange

Called when in-use emphasis changes on an object and the IContainerControl (p. 174) containing the object has an ICnrHandler (p. 98). If the object has just acquired in-use emphasis, *acquired* is set to true.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleInuseChange(IContainerControl* container,	<i>Y</i>	<i>Y</i>	<i>I</i>
Boolean acquired);			

handleOpen

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 (p. 174) containing the object has an ICnrHandler (p. 98).

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleOpen(IContainerControl* container);	<i>Y</i>	<i>Y</i>	<i>Y</i>

handleSelectedChange

Called when selection emphasis changes on an object and the IContainerControl (p. 174) containing the object has an ICnrHandler (p. 98). If the object has just acquired selection emphasis, *acquired* is set to true.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleSelectedChange(IContainerControl* container,	<i>Y</i>	<i>Y</i>	<i>Y</i>
Boolean acquired);			

handleTreeCollapse

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 (p. 174) containing the object has an ICnrHandler (p. 98).

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleTreeCollapse(IContainerControl* container);	<i>Y</i>	<i>Y</i>	<i>Y</i>

handleTreeExpand

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 (p. 174) containing the object has an ICnrHandler (p. 98).

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleTreeExpand(IContainerControl* container);	<i>Y</i>	<i>Y</i>	<i>Y</i>

IContainerObject

helpId

Called when an ICnrHelpEvent (p. 103) is dispatched to ICnrHandler (p. 98). 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.

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

objectCopy

Called when it is necessary to make a copy of an object. This occurs when you call IContainerControl::copyObjectTo (p. 202).

virtual IContainerObject* objectCopy();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

operator ==

If the objects are at the same storage location, true is returned.

Boolean operator ==(const IContainerObject& object);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

Object Allocation

These members provide all object allocation and deallocation for instances of the class. Stack and static instances of this class are not allowed.

operator delete

Deallocates the storage of a class object.

1 void operator delete(void*);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

2 void operator delete(void*, const char* fileName, size_t lineNumber);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

operator new Allocates storage for an object of this class.

1 void* operator new(size_t size, ICnrAllocator& allocator);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

IContainerObject

2	<pre>void* operator new(size_t size, const char* fileName, size_t lineNumber);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
3	<pre>void* operator new(size_t size, const char* fileName, size_t lineNumber, ICnrAllocator& allocator);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
4	<pre>void* operator new(size_t size);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

Object Information

Use these members to get and set the accessible attributes of this class.

Note: Because an object can exist in more than one container, many of these functions accept a container as an optional parameter. If the container is not specified, the function is applied to all containers that this object is in.

disableDataUpdate

Disables user updates of this object.

<pre>virtual IContainerObject& disableDataUpdate(IContainerControl* container = 0);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

disableDrop Prevents any object from being dropped on this object.

<pre>virtual IContainerObject& disableDrop(IContainerControl* container = 0);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

enableDataUpdate

Enables user updates of this object if the container in which this object resides is enabled for direct editing.

<pre>virtual IContainerObject& enableDataUpdate(Boolean enable = true, IContainerControl* container = 0);</pre>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

enableDrop Allows other objects to be dropped on this object.

IContainerObject

<pre>virtual IContainerObject& enableDrop(Boolean enable = true, IContainerControl* container = 0);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

hide Makes this object invisible by filtering it out.

<pre>virtual IContainerObject& hide(IContainerControl* container = 0);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of objects.

icon Returns the icon handle of an object. If this object does not have an icon, the icon handle is zero.

<pre>virtual IPointerHandle icon() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

iconText Returns the text of an object.

<pre>virtual IString iconText() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

isDropOnAble

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.

<pre>virtual Boolean isDropOnAble(IContainerControl* container = 0) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

isInUse 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.

<pre>virtual Boolean isInUse(IContainerControl* container = 0) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

isOpen If this object is open, true is returned.

IContainerObject

```
virtual Boolean  
  isOpen() const;
```

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

isRefreshOn If this object is in a refreshable state, true is returned.

```
virtual Boolean  
  isRefreshOn() const;
```

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

isVisible 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.

```
virtual Boolean  
  isVisible( IContainerControl* container = 0 ) const;
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of columns.

isWriteable Returns true if the data in the object can be edited.

```
virtual Boolean  
  isWriteable( IContainerControl* container = 0 ) const;
```

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

refresh 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 (p. 267) to do so.

If *immediate* is set to true, the painting is done synchronously; that is, all container painting is completed before this function returns to the caller. If *immediate* is false, the container painting occurs asynchronously, and painting may not be completed when the function returns to the caller.

```
virtual IContainerObject&  
  refresh( IContainerControl* container = 0,  
          Boolean immediate = false );
```

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

removeInUse Removes the in-use emphasis for this object.

```
virtual IContainerObject&  
  removeInUse( IContainerControl* container = 0 );
```

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

setClosed Removes the open emphasis in all containers.

IContainerObject

	virtual IContainerObject& setClosed();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
setIcon	Sets a new icon for an object.			
1	virtual IContainerObject& setIcon(const IPointerHandle& iconHandle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
2	virtual IContainerObject& setIcon(const IResourceId& iconId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
3	virtual IContainerObject& setIcon(unsigned long iconId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
setIconText	Sets the new text for an object.			
1	virtual IContainerObject& setIconText(const IString& iconText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
2	virtual IContainerObject& setIconText(const char* iconText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
3	virtual IContainerObject& setIconText(const IResourceId& iconTextId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
setInUse	Sets the in-use emphasis for this object.			
	virtual IContainerObject& setInUse(Boolean inUse = true, IContainerControl* container = 0);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
Win	The native Windows containers (that is, containers constructed without the pmCompatible style) do support the in-use emphasis but require that the icon have transparency. The hashed background is only displayed where the object's icon has transparency.			
setOpen	Sets the open emphasis. This function can be overridden by the application to provide specialized behavior for an enter event at the object level. By default, this function calls setInUse to set the in-use emphasis.			

IContainerObject

```
virtual IContainerObject&
  setOpen( Boolean open = true );
```

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

setRefreshOff

Prevents refreshing of this object.

```
virtual IContainerObject&
  setRefreshOff();
```

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

setRefreshOn

Allows refreshing of this object.

```
virtual IContainerObject&
  setRefreshOn( Boolean refreshOn = true );
```

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

show

Makes this object visible.

```
virtual IContainerObject&
  show( Boolean show = true,
        IContainerControl* container = 0 );
```

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



The native Windows containers (that is, containers constructed without the pmCompatible style) do not support filtering of objects.

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Attributes

These members implement the class.

baseRecord Returns the address of the real base of this object.

IContainerObject

IMiniCnrRecord* baseRecord();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------------------------------	-----------------	----------------	-------------------

decrementUseCount

Decreases the count of containers having this object.

virtual IContainerObject& decrementUseCount();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

incrementUseCount

Increases the count of containers having this object.

virtual IContainerObject& incrementUseCount();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

initialize

Initializes an object after construction.

virtual IContainerObject& initialize();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

isAttribute

If the specified attribute is set to on, true is returned.

virtual Boolean isAttribute(unsigned long attribute, IContainerControl* container) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

setAttributes 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.

virtual IContainerObject& setAttributes(unsigned long attributeTurnedOff, unsigned long attributeTurnedOn, IContainerControl* container = 0);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

setBase

Stores the address of the real base of this object.

virtual IContainerObject& setBase(const IMiniCnrRecord* baseRecord);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

setEmphasis 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.

IContainerObject

```
virtual IContainerObject&
    setEmphasis( unsigned long emphasisAttribute,
                  Boolean turnOn = true,
                  IContainerControl* container = 0 );
```

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

useCount Returns the number of containers having this object.

```
unsigned long
    useCount() const;
```

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

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Emphasis

```
Emphasis {
    none = 0,
    censored = 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.

censored

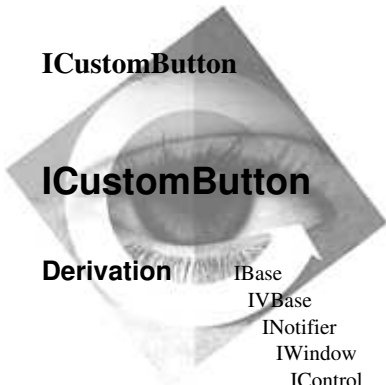
Provides a visible indication that the object contains the cursor.

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.



ICustomButton

ICustomButton

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IButton
ICustomButton

Inherited By

IAnimatedButton
IToolBarButton

Header File

icustbut.hpp

Members

Member	Page	Member	Page
Constructor	272	latch	274
autoLatch	278	latchable	278
backgroundColor	271	latchedBackgroundColor	271
calcMinimumSize	277	latchedForegroundColor	271
classDefaultStyle	278	latchId	277
convertToGUIStyle	274	resetLatchedBackgroundColor	271
defaultStyle	275	resetLatchedForegroundColor	271
disableAutoLatch	273	setDefaultStyle	275
disableLatching	273	setLatchedBackgroundColor	272
enableAutoLatch	273	setLatchedForegroundColor	272
enableLatching	273	setUserData	275
isAutoLatchEnabled	273	unlatch	274
isLatched	273	userData	275
isLatchedBackgroundColorHalfTone	271	~ICustomButton	272
isLatchingEnabled	274		

The ICustomButton class creates and manages the custom button window. Custom buttons allow the application to customize the button appearance by providing an ICustomButtonDrawHandler to draw the button.

Public Functions

Colors

Use these members to set or query the color attributes of a custom button.

ICustomButton

backgroundColor

Returns the color used to paint the background of the custom button when it is in the default state.

virtual IColor backgroundColor() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

isLatchedBackgroundColorHalftone

Returns true if the background is a halftone when the custom button is in the latched state.

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

latchedBackgroundColor

Returns the color used to paint the background of the custom button when it is in the latched state.

virtual IColor latchedBackgroundColor() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

latchedForegroundColor

Returns the color used to paint the foreground of the custom button when it is in the latched state.

virtual IColor latchedForegroundColor() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

resetLatchedBackgroundColor

Resets the latched background color by undoing a previous set.

virtual ICustomButton& resetLatchedBackgroundColor();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

resetLatchedForegroundColor

Resets the latched foreground color by undoing a previous set.

virtual ICustomButton& resetLatchedForegroundColor();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

ICustomButton

setLatchedBackgroundColor

Sets the color used to paint the background of the button when it is in the latched state.

```
virtual ICustomButton&
    setLatchedBackgroundColor( const IColor& color,
                               Boolean halftone = true );
```

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

setLatchedForegroundColor

Sets the color used to paint the foreground of the button when it is in the latched state.

```
virtual ICustomButton&
    setLatchedForegroundColor( const IColor& color );
```

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

Constructors

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

ICustomButton

1

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

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

Creates an ICustomButton with the specified window ID, parent and owner windows, screen position and size, and window style.

Exceptions	
InvalidParameter	The parent window pointer is invalid.

2

```
ICustomButton( const IWindowHandle& handle );
```

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

Creates an ICustomButton object for the specified button control that has a user-button style.

~ICustomButton

ICustomButton

```
virtual  
~ICustomButton();
```

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

Latch Support

Use these members to manage the latched state of a custom button.

disableAutoLatch

Disables the autoLatch style.

```
virtual ICustomButton&  
disableAutoLatch();
```

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

disableLatching

Sets the style of the button so that the user cannot latch or unlatch it.

```
virtual ICustomButton&  
disableLatching();
```

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

enableAutoLatch

Sets the style of the button so that other custom buttons in the same group that have the autoLatch style are automatically unlatched when the user latches the button.

```
virtual ICustomButton&  
enableAutoLatch( Boolean enable = true );
```

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

enableLatching

Sets the style of the button so that the user can latch or unlatch it.

```
virtual ICustomButton&  
enableLatching( Boolean enable = true );
```

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

isAutoLatchEnabled

Returns true if the autoLatch style is set for the button.

```
virtual Boolean  
isAutoLatchEnabled() const;
```

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

isLatched

Returns true if the button is in a latched state.

ICustomButton

```
virtual Boolean  
    isLatched() const;
```

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

isLatchingEnabled

Returns true if the latchable style is set for the button.

```
virtual Boolean  
    isLatchingEnabled() const;
```

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

latch

Puts the button in a latched state.

```
virtual ICustomButton&  
    latch( Boolean latched = true,  
          Boolean refresh = true );
```

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

unlatch

Puts the button in the unlatched (default) state.

```
virtual ICustomButton&  
    unlatch();
```

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

Styles

Use these members to set and query ICustomButton styles. You can use these styles with the styles defined by the following nested classes:

- IWindow Styles (Vol. II)
- IControl Styles (Vol. II)
- IButton Styles (Vol. II)

If the latchable style is set for the button but not the autoLatch style, then the user can latch and unlatch the button directly. If the autoLatch style is set for the button but not the latchable style, then the user can latch the button but cannot unlatch it directly. The user must latch another custom button that has the autoLatch style to unlatch the previous button. If both the latchable and autoLatch styles are set for the button, then the user can latch and unlatch the button directly. Also, the button automatically unlatches itself when the user latches another custom button that has the autoLatch style.

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform.

Extended styles, those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

ICustomButton

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

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

defaultStyle Returns the default style. The default style is classDefaultStyle unless you have changed the style using setDefaultStyle.

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

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

setDefaultStyle

Sets the default style for all subsequent custom buttons.

style Use the styles provided by ICustomButton::Styles to specify the default style.

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

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

User Data

Use these members to set or query user data for custom button objects.

setUserData Stores an unsigned long value that represents special data to be associated with the button object.

```
ICustomButton&  
    setUserData( unsigned long data );
```

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

userData Returns an unsigned long value that represents special data associated with the button object. This data can be used to store information about special drawing for the button.

```
unsigned long  
    userData() const;
```

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

Inherited Public Functions

IButton		
allowsMouseClickedFocus	disableMouseClickedFocus	highlight
backgroundColor	enableMouseClickedFocus	hiliteBackgroundColor

ICustomButton

IButton		
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Constructors

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

ICustomButtonICustomButton();

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

ICustomButton

Layout Support

These members provide layout support for custom buttons.

calcMinimumSize

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

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

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

These INotificationId strings are used for all notifications that ICustomButton provides to its observers.

latchId

This notification identifier is provided to observers when the user latches or unlatches the button. A Boolean value is provided in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is true if the button is currently in the latched state.

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
latchId;	<i>Y</i>	<i>Y</i>	<i>N</i>

Styles

Use these members to set and query ICustomButton styles. You can use these styles with the styles defined by the following nested classes:

- IWindow Styles (Vol. II)
- IControl Styles (Vol. II)
- IButton Styles (Vol. II)

ICustomButton

If the latchable style is set for the button but not the autoLatch style, then the user can latch and unlatch the button directly. If the autoLatch style is set for the button but not the latchable style, then the user can latch the button but cannot unlatch it directly. The user must latch another custom button that has the autoLatch style to unlatch the previous button. If both the latchable and autoLatch styles are set for the button, then the user can latch and unlatch the button directly. Also, the button automatically unlatches itself when the user latches another custom button that has the autoLatch style.

autoLatch Unlatches automatically other custom buttons in the same group when the user latches button. This style also allows the user to latch the button when clicking on the button with mouse button 1.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
autoLatch;	<i>Y</i>	<i>Y</i>	<i>N</i>

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

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

latchable Allows the user to latch or unlatch the custom button when clicking on the button with mouse button 1.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
latchable;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

IButton		
buttonClickId	noPointerFocus	
ITextControl		
textId		
IControl		
group	tabStop	

ICustomButton

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

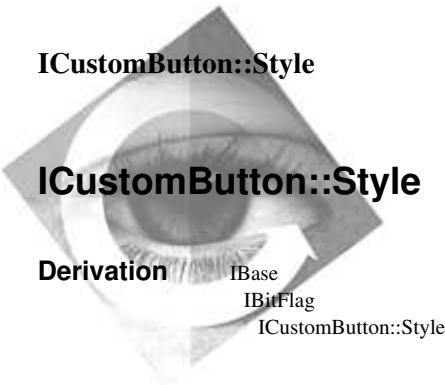
Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ICustomButton contains the following nested classes:

ICustomButton::Style (see page 280)



ICustomButton::Style

ICustomButton::Style

Derivation IBase
IBitFlag
ICustomButton::Style

Inherited By None.

Header File icustbut.hpp

The nested class ICustomButton::Style provides a set of valid styles for the static member functions of the class ICustomButton.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ICustomButtonDrawEvent

Derivation

```
IBase
  IVBase
    IEvent
      IControlEvent
        ICustomButtonDrawEvent
```

Header File `icustbev.hpp`

The `ICustomButtonDrawEvent` class provides information about a custom button event for `ICustomButtonDrawHandler` event handling functions.

Constructors

ICustomButtonDrawEvent

Creates an ICustomButtonDrawEvent from the specified IEvent.

Creates an ICustomButtonDrawEvent from the specified ICustomButtonDrawEvent.

ICustomButtonDrawEvent

3	ICustomButtonDrawEvent(const IEvent& event, const ISize& size);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	------------------------	-----------------------	--------------------------

Creates an ICustomButtonDrawEvent from the specified IEvent and graphic context dimensions. Use this constructor if you want to create a bitmap image of the button. When you use this constructor, the graphic context returned by graphicContext (p. 283) is a memory graphic context. After drawing of the button is complete, you can use IGBitmap (Vol. IV) to capture the bitmap image.

~ICustomButtonDrawEvent

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

Custom Button

Use these members to query information about the custom button.

customButton

Returns a pointer to the ICustomButton object that is to be drawn.

virtual ICustomButton*	customButton() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
------------------------	-----------------------	------------------------	-----------------------	--------------------------

Drawing

Use these members to query or modify the event-drawing information.

buttonState Returns the state that describes how the button is to be drawn.

ButtonState	buttonState() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
-------------	----------------------	------------------------	-----------------------	--------------------------

drawDown Returns true if the button is to be drawn in the down state.

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

drawingArea Returns the current drawing area rectangle to be used when drawing the button.

IRectangle	drawingArea() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
------------	----------------------	------------------------	-----------------------	--------------------------

ICustomButtonDrawEvent

drawLatched Returns true if the button is to be drawn in the latched state.

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

drawUp Returns true if the button is to be drawn in the up state.

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

graphicContext

Returns a graphic context to be used when drawing the button.

IGraphicContext&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
graphicContext() const;	Y	Y	N

isButtonEnabled

Returns true if the button is to be drawn in the enabled state.

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

setDrawingArea

Sets the current drawing area rectangle to be used when drawing the button.

virtual ICustomButtonDrawEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDrawingArea(const IRectangle& rectangle);	Y	Y	N

Inherited Public Functions

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner

ICustomButtonDrawEvent

IEvent		
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

ButtonState ButtonState {
 buttonUp,
 buttonDown,
 buttonLatched
 };

Enumeration used to describe the state to be used to draw the button:

buttonUp

The button is to be drawn in the up state.

buttonDown

The button is to be drawn in the down state.

buttonLatched

The button is to be drawn in the latched state.

ICustomButtonDrawHandler



ICustomButtonDrawHandler

Derivation

- IBase
- IVBase
- IHandler
- ICustomButtonDrawHandler

Inherited By None.

Header File icustbhd.hpp

Members	Member	Page	Member	Page
	Constructor	285	drawButton	287
	dispatchHandlerEvent	286	drawDisabledEmphasis	287
	drawBackground	286	drawForeground	287
	drawBorder	287	~ICustomButtonDrawHandler	285

The ICustomButtonDrawHandler class is processes ICustomButton drawing events.

Note: Attach objects of this class only to ICustomButton objects. Attaching an ICustomButtonDrawHandler to the owner of an ICustomButton will not work correctly.

Public Functions

Constructors

You can construct and destruct objects of this class.

ICustomButtonDrawHandler

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

The only way to construct objects of this class is with the default constructor.

~ICustomButtonDrawHandler

virtual ~ICustomButtonDrawHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

ICustomButtonDrawHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

The User Interface Class Library dispatches events that have been sent or posted to a window to the handlers attached to that window. It does this by calling the event-dispatching function of the handler objects. An ICustomButtonDrawHandler object processes only custom button draw events.

dispatchHandlerEvent

Calls the appropriate virtual function or functions to process a custom button draw event.

virtual Boolean
dispatchHandlerEvent(IEvent& event);

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

Event Processing

Event-processing members provide information about handling events.

drawBackground

Called to draw the background of the button. You can override this function to change the way the button's background is drawn.

The default behavior is to fill the current drawing area with the background color or latched background color.

ICustomButtonDrawHandler

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
drawBackground(ICustomButtonDrawEvent& event);	Y	Y	N

drawBorder Called to draw the border of the button. The default implementation draws the border and removes the area painted from the button drawing area. Uses the margin option to specify the amount of white space that to be left around the button. You can override this function to change the way the button border is drawn.

Note: If the button's drawing area rectangle is not reduced (by removing the area occupied by the border), then the background and foreground of the button can paint over the border.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
drawBorder(ICustomButtonDrawEvent& event,	Y	Y	N
ISize margin = ISize ());			

drawButton Called when the button needs to be redrawn on the screen. This can occur when the button has just become visible or if the state of the button has changed.

You can override this function to change the way the button is displayed on the screen. The default implementation calls the functions drawBorder, drawBackground, drawForeground, and (if disabled) drawDisabledEmphasis, and then returns true.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
drawButton(ICustomButtonDrawEvent& event);	Y	Y	N

drawDisabledEmphasis

Called to draw disabled emphasis for the button. You can override this function to change the way the button's disabled emphasis is drawn.

The default behavior is to use a halftone in the current drawing area.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
drawDisabledEmphasis(ICustomButtonDrawEvent& event);	Y	Y	N

drawForeground

Called to draw the foreground of the button. You can override this function to change the way the button's foreground is drawn.

The default behavior is to draw the custom button text with the foreground or latched foreground color.

ICustomButtonDrawHandler

```
virtual void
drawForeground( ICustomButtonDrawEvent& event );
```

Win
Y

PM
Y

Motif
N

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEAcknowledgeEvent

Derivation

```

IBase
IVBase
IEvent
IDDEEvent
IDDEAcknowledgeEvent
  
```

Inherited By

```

IDDEAcknowledgeExecuteEvent
IDDEAcknowledgePokeEvent
IDDEClientAcknowledgeEvent
IDDEServerAcknowledgeEvent
  
```

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	289	isMessageUnderstood	290
	applicationSpecificData	290	transactionType	291
	isAckPositive	290	~IDDEAcknowledgeEvent	290
	isApplicationBusy	290		

The IDDEAcknowledgeEvent class is the base class for the Dynamic Data Exchange (DDE) acknowledge-event information classes.

An object of this class or one of its derived classes is created when an IDDEClientConversation object or IDDETopicServer object needs to pass acknowledgment information to the client or server application. You do not construct objects of this class. See IDDEClientConversation (p. 309) and IDDETopicServer (p. 358) for information about those classes.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer or IDDEClientConversation objects.

IDDEAcknowledgeEvent

```
IDDEAcknowledgeEvent( const IEvent& ddeEvent );
```

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

IDDEAcknowledgeEvent

~IDDEAcknowledgeEvent

~IDDEAcknowledgeEvent();

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

Event Information

Use these members to query information about an acknowledgment.

applicationSpecificData

Returns any application-specific data that has been provided by the conversing application.

unsigned char
applicationSpecificData() const;

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

isAckPositive

Returns true if the acknowledgment is positive; otherwise, it returns false.

Boolean
isAckPositive() const;

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

isApplicationBusy

Returns true if the application busy flag is on; otherwise, it returns false. This is only set for a negative acknowledgment.

Boolean
isApplicationBusy() const;

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

isMessageUnderstood

Returns false if the message-not-understood flag is on; otherwise, it returns true. This is only set for a negative acknowledgment.

Boolean
isMessageUnderstood() const;

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

IDDEAcknowledgeEvent

Inherited Public Functions

IDDEEvent		
format	item	

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Information

Use these members to query information about an acknowledgment.

transactionType

Returns the event ID.

unsigned short	<u>Win</u>	<u>PM</u>	<u>Motif</u>
transactionType() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

IDDEAcknowledgeEvent

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEAcknowledgeExecuteEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDDEEvent
        IDDEAcknowledgeEvent
          IDDEAcknowledgeExecuteEvent
  
```

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page
	Constructor	293
	commands	294
	~IDDEAcknowledgeExecuteEvent	294

The IDDEAcknowledgeExecuteEvent class provides event information to a client application regarding an acknowledgment of an executeCommands request.

An object of this class is created when an IDDEClientConversation object needs to pass information to the client application about an acknowledgment resulting from an IDDEClientConversation::executeCommands request. You do not construct objects of this class. See IDDEClientConversation (p. 309) and executeCommands (p. 315) for information about that class and function.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDEClientConversation objects.

IDDEAcknowledgeExecuteEvent

```
IDDEAcknowledgeExecuteEvent( const IEvent& ddeEvent );
```

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

IDDEAcknowledgeExecuteEvent

~IDDEAcknowledgeExecuteEvent

```
~IDDEAcknowledgeExecuteEvent();
```

Win

PM

Motif

Y

Y

N

Event Information

Use these members to query information about the command string you sent to the DDE server application.

commands Returns the command string that the client asked the server application to execute.

Note: The IDDEAcknowledgeExecuteEvent::commands function's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and function.

```
IString  
  commands() const;
```

Win

PM

Motif

Y

Y

N

Inherited Public Functions

IDDEAcknowledgeEvent		
applicationSpecificData	isAckPositive	isApplicationBusy

IDDEEvent		
format	item	

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

IDDEAcknowledgeExecuteEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEAcknowledgeEvent		
transactionType		

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IDDEAcknowledgePokeEvent

IDDEAcknowledgePokeEvent

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IEvent
    IEvent --> IDDEEvent
    IDDEEvent --> IDDEAcknowledgeEvent
    IDDEAcknowledgeEvent --> IDDEAcknowledgePokeEvent
```

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page
	Constructor	296
	pokedData	297
	~IDDEAcknowledgePokeEvent	297

The IDDEAcknowledgePokeEvent class provides event information to a client application regarding an acknowledgment to a IDDEClientConversation::pokeData request.

An object of this class is created when an IDDEClientConversation object needs to pass information to the client application about an acknowledgment resulting from a pokeData request. You do not construct objects of this class. See IDDEClientConversation (p. 309) and pokeData (p. 316) for information about that class and function.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDEClientConversation objects.

IDDEAcknowledgePokeEvent

```
IDDEAcknowledgePokeEvent( const IEvent& ddeEvent );
```

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

IDDEAcknowledgePokeEvent

~IDDEAcknowledgePokeEvent

~IDDEAcknowledgePokeEvent();

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

Event Information

Use these members to query information about the data you sent to the DDE server application.

pokedData Returns the data that the client asked the server application to set.

Note: The IDDEAcknowledgePokeEvent::pokedData function's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and function.

IString
pokedData() const;

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

Inherited Public Functions

IDDEAcknowledgeEvent		
applicationSpecificData	isAckPositive	isApplicationBusy

IDDEEvent		
format	item	

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

IDDEAcknowledgePokeEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEAcknowledgeEvent		
transactionType		

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEActiveServer

Derivation IBase
IDDEActiveServer

Inherited By None.

Header File iddecset.hpp

Members	Member	Page	Member	Page
	Constructor	299	topic	300
	application	300	~IDDEActiveServer	299
	isCaseSensitive	300		

The IDDEActiveServer class holds information about a Dynamic Data Exchange (DDE) server application and a particular topic the server application supports.

Objects of this class are created to provide information about possible conversations when one of the two broadcast functions, IDDEClientConversation::supportedTopics or IDDEClientConversation::supportingApplications, is called.

IDDEClientConversation objects create objects of this class, which are not intended to be created by application programmers. See IDDEClientConversation (p. 309), supportedTopics (p. 310), or supportingApplications (p. 310) for more information.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDEClientConversation objects.

IDDEActiveServer

IDDEActiveServer(const char* applicationName,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	const char* topicName,	Y	Y	N
	Boolean caseSensitive);			

~IDDEActiveServer

IDDEActiveServer

<code>~IDDEActiveServer();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Server Attributes

Use these members to query information about a DDE server application.

application Returns the name of the server application.

<code>NSString application() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

isCaseSensitive

Returns true if the server enforces case sensitivity; otherwise, it returns false.

<code>Boolean isCaseSensitive() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

topic Returns the name of the topic supported by the server.

<code>NSString topic() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IBase		
<code>asDebugInfo</code>	<code>messageFile</code>	<code>setMessageFile</code>
<code>asString</code>	<code>messageText</code>	<code>version</code>

Inherited Protected Data

IBase		
<code>recoverable</code>	<code>unrecoverable</code>	



IDDEActiveServerSet

Derivation ISet
 IDDEActiveServerSet

Inherited By None.

Header File iddecset.hpp

Members	Member	Page
	Constructor	301
	~IDDEActiveServerSet	302

The IDDEActiveServerSet class is a set created using the ISet template class. The set contains pointers to IDDEActiveServer objects. See IDDEActiveServer (p. 299) for information about that class.

An application creates an object of this class to use with the IDDEClientConversation::supportingApplications or IDDEClientConversation::supportedTopics function to hold the current set of active servers. If you remove individual elements of this set, delete the IDDEActiveServer object to which the element points or the memory is not recovered. See supportingApplications (p. 310) and supportedTopics (p. 310) for information about those functions.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDDEActiveServerSet

IDDEActiveServerSet();

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

You can construct an object of this class using the default constructor, which does not accept any arguments.

IDDEActiveServerSet

~IDDEActiveServerSet

This function removes all the elements from the set and deletes them.

```
~IDDEActiveServerSet();
```

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



IDDEBeginEvent

Derivation IBase
 IVBase
 IEvent
 IDDEBeginEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	303	topic	304
	application	304	~IDDEBeginEvent	303
	setCaseSensitive	304		

The IDDEBeginEvent class provides event information to a server application when a client has asked the server to begin a conversation.

An object of this class is created when an IDDETopicServer object needs to pass an acceptConversation request to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) and acceptConversation (p. 365) for information about that class and function.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDEBeginEvent

```
IDDEBeginEvent( const IEvent& ddeBeginEvent );
```

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

~IDDEBeginEvent

IDDEBeginEvent

```
~IDDEBeginEvent();
```

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

Event Attributes

Use these members to query information about the requested conversation and to set the case sensitivity of the conversation, if it is accepted.

application Returns the name of the requested application.

```
IString  
application() const;
```

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

setCaseSensitive

Sets the case-sensitive flag to true. The server application should set this flag if it enforces case sensitivity.

```
IDDEBeginEvent&  
setCaseSensitive( Boolean caseSensitive );
```

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

topic Returns the name of the requested topic.

```
IString  
topic() const;
```

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

Inherited Public Functions

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

IVBase		
asDebugInfo	asString	

IDDEBeginEvent

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEClientAcknowledgeEvent

IDDEClientAcknowledgeEvent

Derivation

- IBase
- IVBase
- IEvent
- IDDEEvent
- IDDEAcknowledgeEvent
- IDDEClientAcknowledgeEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	306	isAckToRequestData	307
	isAckToBeginHotLink	307	~IDDEClientAcknowledgeEvent	307
	isAckToEndHotLink	307		

The IDDEClientAcknowledgeEvent class provides event information to a client application regarding an acknowledgment from a server.

An object of this class or one of its derived classes is created when an IDDEClientConversation object needs to pass information about an acknowledgment to the client application. You do not construct objects of this class. See IDDEClientConversation (p. 309) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDEClientConversation objects.

IDDEClientAcknowledgeEvent

```
IDDEClientAcknowledgeEvent( const IEvent& ddeEvent );
```

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

IDDEClientAcknowledgeEvent

~IDDEClientAcknowledgeEvent

~IDDEClientAcknowledgeEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to query information about the type of the event that the acknowledgment is for.

isAckToBeginHotLink

Returns true if the acknowledgment is in response to IDDEClientConversation::beginHotLink being called; otherwise, it returns false.

See beginHotLink (p. 314) for information about that member.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isAckToBeginHotLink() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isAckToEndHotLink

Returns true if the acknowledgment is in response to IDDEClientConversation::endHotLink being called; otherwise, it returns false.

See endHotLink (p. 315) for information about that member.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isAckToEndHotLink() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isAckToRequestData

Returns true if the acknowledgment is in response to IDDEClientConversation::requestData being called; otherwise, it returns false.

See requestData (p. 316) for information about that member.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isAckToRequestData() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDDEClientAcknowledgeEvent

IDDEAcknowledgeEvent		
applicationSpecificData	isAckPositive	isApplicationBusy

IDDEEvent		
format	item	

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEAcknowledgeEvent		
transactionType		

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEClientConversation

Derivation

- IBase
- IVBase
- IHandler
- IDDEClientConversation

Inherited By None.

Header File iddeccnv.hpp

Members				
	Member	Page	Member	Page
	Constructor	311	handleData	318
	acknowledged	319	handleInitiateAck	318
	application	312	handleTerminate	318
	begin	313	hotLinkCount	314
	beginHotLink	314	hotLinkInform	320
	clientHandle	312	hotLinks	314
	conversationEnded	319	inConversation	312
	data	319	isCaseSensitive	312
	dispatchHandlerEvent	317	outstandingTransactionCount	312
	end	313	pokeAcknowledged	320
	endAllHotLinks	321	pokeData	316
	endHotLink	315	requestData	316
	endHotLinks	315	supportedTopics	310
	executeAcknowledged	320	supportingApplications	310
	executeCommands	315	topic	312
	findTransaction	321	~IDDEClientConversation	311
	handleAck	318		

The IDDEClientConversation class adds Dynamic Data Exchange (DDE) client function to an application.

Create an object of this class for each conversation you want to initiate with a server application. The object manages all window, shared memory, and atom table processing. When a conversation with a server ends, the IDDEClientConversation object can initiate another conversation with that server or any other server. An object of this class can find all topics supported by all DDE server applications on the system.

This class uses a window to communicate; therefore, window message processing must occur. This means that ICurrentThread::processMsgs must be called. There are several ways for this to occur. Normally, this is accomplished by calling

IDDEClientConversation

IApplication::current().run(). See processMsgs (Vol. II), current (Vol. II), and run (Vol. II) for information about those members.

Public Functions

Broadcasts

Use these members to query information about the topics supported by the currently active DDE server applications.

supportedTopics

Adds IDDEActiveServer objects that represent topics supported by DDE server applications to the IDDEActiveServerSet passed in as the first argument. If the applicationName argument is specified, only the topics supported by that application are added to the set. The default is to add all topics supported by all server applications.

Note: Include the iddecset.hpp header file when you use this function.

See IDDEActiveServer (p. 299) and IDDEActiveServerSet (p. 301) for information about those classes.

```
virtual IDDEClientConversation&
    supportedTopics( IDDEActiveServerSet& activeServerSet,
                    const char* applicationName = 0 );
```

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

supportingApplications

Adds IDDEActiveServer objects that represent DDE server applications that support the specified topic to the IDDEActiveServerSet passed in as the first argument.

Note: Include the iddecset.hpp header file when you use this function.

See IDDEActiveServer (p. 299) and IDDEActiveServerSet (p. 301) for information about those classes.

```
virtual IDDEClientConversation&
    supportingApplications(
        IDDEActiveServerSet& activeServerSet,
        const char* topicName );
```

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

Exceptions	
InvalidParameter	The topicName parameter is NULL.

IDDEClientConversation

Constructors

You can construct objects of this class in two ways. Both constructors have a Boolean argument, `useEventThread`. This argument is optional but is also highly recommended if the application must do any extensive processing or interacts with the user during any of the callback functions. Specifying `true` allows `IDDEClientConversation` to create a secondary thread to process incoming events. This prevents problems with window message processing because the thread is created without a message queue. If you specify `false`, no secondary thread is created, and you should return promptly from all callback functions.

IDDEClientConversation

	<u>Win</u>	<u>PM</u>	<u>Motif</u>
1 IDDEClientConversation(Boolean useEventThread = true);	Y	Y	N

This constructor initializes the object but does not start a conversation.

Exceptions	
IOutfOfSystemResource	The operating system's request for either an event semaphore or a queue failed. See the text of the exception for further information.

2	IDDECClientConversation(const char* applicationName, const char* topicName, Boolean useEventThread = true);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	---	-----------------	----------------	-------------------

This constructor accepts the following arguments and attempts to begin a conversation with the requested server application on the requested topic:

- The name of the server application with which a conversation is desired
- The name of the topic about which a conversation is desired

Exceptions	
IOOutOfSystemResource	The operating system's request for either an event semaphore or a queue failed. See the text of the exception for further information.

~IDDEClientConversation

The destructor ends the conversation with the DDE server application if there is one active.

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

IDDEClientConversation

Conversation Information

Use these members to obtain the attributes of this object, including information about the conversation if there is one currently active.

application Returns the name of the server application with which the client is conversing.

IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
application() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

clientHandle Returns the window handle of the client conversation. This is provided, along with the IDDEClientConversation::begin member, to allow an object of this class to converse with a server without engaging in the normal conversation initialization.

See begin (p. 313) for information about that member.

IWindowHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
clientHandle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

inConversation

Returns true if the client is currently in conversation with a server; otherwise, it returns false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
inConversation() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isCaseSensitive

Returns true if the server has indicated it enforces case sensitivity; otherwise, it returns false.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isCaseSensitive() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

outstandingTransactionCount

Returns the number of transactions initiated by the client to which the server has not responded.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
outstandingTransactionCount() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

topic Returns the name of the topic about which the client is conversing.

IDDEClientConversation

CString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
topic() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Conversation Initiation and Termination

Use these members to begin and end conversations with DDE server applications.

begin

Attempts to initiate a conversation with the requested server application on the requested topic. The primary version of this overloaded member accepts the name of the requested application and topic as input and returns true if the conversation is successfully begun; otherwise, it returns false. The version that accepts a server handle is provided along with the IDDEClientConversation::clientHandle member to allow an object of this class to converse with a server without engaging in the normal conversation initialization.

See clientHandle (p. 312) for information about that function.

1	virtual IDDEClientConversation& begin(const IWindowHandle& serverHandle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidParameter	The server window handle is not valid.
InvalidRequest	The object is already in conversation with another application.

2	virtual Boolean begin(const char* applicationName, const char* topicName);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidParameter	The applicationName or topicName parameter is NULL or has a zero length.
InvalidRequest	The object is already in conversation with another application.
IAccessError	The operating system's request to begin a conversation failed.

end

Ends the current conversation.

virtual IDDEClientConversation& end();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidRequest	The object is not in conversation with a server application.

IDDEClientConversation

Hot Link Information

Use these members to obtain information about the hot links currently active in this object.

hotLinkCount

Returns a count of the number of hot links.

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

hotLinks

Adds all item and format pairs for which hot links have been established to the IDDEClientHotLinkSet passed in.

Note: Include the iddecset.hpp header file when you use this function.

See IDDEClientHotLinkSet (p. 329) for information about that class.

IDDEClientConversation& hotLinks(IDDEClientHotLinkSet& hotLinkSet);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Transactions

Use these members to send transactions to the DDE server application. Most of them accept a string that identifies the format in which the data is rendered. The default is IDDE::textFormat, which specifies text format. Formats are standardized by various vendors, who describe them in their operating system or application documentation.

beginHotLink Requests the server to send a notification every time the value of the item changes. If the *sendData* argument is true, the data is sent with the notification in the requested format. If the *pacing* argument is true, the server is asked to wait for an acknowledgment from the client before sending a subsequent notification.

This function accepts a format string that identifies the format in which the data is rendered. The default is IDDE::textFormat, which specifies the text format.

Formats are standardized by various vendors, who describe them in their operating system or application documentation.

virtual IDDEClientConversation& beginHotLink(const char* item, IDDE::Format format = IDDE::textFormat, Boolean sendData = true, Boolean pacing = false);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

IDDEClientConversation

Exceptions	
InvalidRequest	The object is not in conversation with a server application, or a hot link for this item and format is already active or pending.
InvalidParameter	The item or format parameter is NULL or has a zero length.
IAccessError	The operating system's request to post the DDE message to the server application failed.

endHotLink Requests the server to end a hot link on the specified item and format. This member accepts a format string that identifies the format in which the data is rendered. The default is IDDE::textFormat, which specifies the text format.

Formats are standardized by various vendors, who describe them in their operating system or application documentation.

```
virtual IDDEClientConversation&
    endHotLink( const char* item,
               IDDE::Format format = IDDE::textFormat );
```

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

Exceptions	
InvalidRequest	The object is not in conversation with a server application, there is no hot link for this item and format, or an endHotLink request for this item and format is pending.
InvalidParameter	The item or format parameter is NULL or has a zero length.
IAccessError	The operating system's request to post the DDE message to the server application failed.

endHotLinks Requests the server to end one or more hot links. If the *item* argument is specified, the server is requested to end all hot links on the item in all formats. If no arguments are provided, the server is requested to end all hot links on all items in all formats.

```
virtual IDDEClientConversation&
    endHotLinks( const char* item = 0 );
```

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

Exceptions	
InvalidRequest	The object is not in conversation with a server application.

executeCommands

Requests the server to execute one or more commands.

```
virtual IDDEClientConversation&
    executeCommands( const void* commands,
                    unsigned long commandLength );
```

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

IDDEClientConversation

Exceptions	
IInvalidRequest	The object is not in conversation with a server application.
IInvalidParameter	The commands or commandLength parameter is NULL.
IAccessError	The operating system request to post the DDE message to the server application failed.

pokeData

Requests the server to set the specified item to the value of the data provided. This function accepts a format string that identifies the format in which the data is rendered. The default is IDDE::textFormat, which specifies the text format.

Formats are standardized by various vendors, who describe them in their operating system or application documentation.

```
virtual IDDEClientConversation&
    pokeData( const char* item,
              const void* data,
              unsigned long dataLength,
              IDDE::Format format = IDDE::textFormat );
```

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

Exceptions	
IInvalidRequest	The object is not in conversation with a server application.
IInvalidParameter	Either the item, format, or data parameter is NULL, or the item or format parameter has a zero length.
IAccessError	The operating system's request to post the DDE message to the server application failed.

requestData

Requests the value of the item rendered in the specified format from the server. This function accepts a format string that identifies the format in which the data is rendered. The default is IDDE::textFormat, which specifies the text format.

Formats are standardized by various vendors, who describe them in their operating system or application documentation.

```
virtual IDDEClientConversation&
    requestData( const char* item,
                IDDE::Format format = IDDE::textFormat );
```

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

Exceptions	
IInvalidRequest	The object is not in conversation with a server application.
IInvalidParameter	The item or format parameter is NULL or has a zero length.
IAccessError	The operating system's request to post the DDE message to the server application failed.

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members process events sent by the DDE server application. The dispatchHandlerEvent function is called whenever an event is sent to this object, and it calls one of the other appropriate members in this object to actually process the event.

Typically, you do not need to override any of these members. If you want to provide some additional behavior before or after the event is processed, make sure that you call the IDDEClientConversation version of the member or unpredictable results can occur and the conversation may not be able to process further events from the DDE server application. You may need to override one of these members if the server application sends a DDE message not supported by the DDE protocol at that particular point in the conversation. You may need to do this because the IDDEClientConversation object throws these invalid events away.

dispatchHandlerEvent

Calls the appropriate virtual member if a DDE client conversation event is found.

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

Win	PM	Motif
Y	Y	N

Exceptions	
IOutOfSystemResource	The operating system's request to write to a queue failed. See the text of the exception for further information.

IDDEClientConversation

Exceptions	
IAccessError	This function has been called recursively, possibly violating DDE synchronization rules. Either construct this object with useEventThread=true or avoid displaying dialog boxes in the event callback functions.

handleAck Handles acknowledgments from server applications.

virtual void handleAck(const IEvent& ackEvent);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

handleData Handles data sent from server applications.

virtual void handleData(const IEvent& dataEvent);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

handleInitiateAck

Handles acknowledgments from server applications to begin requests.

virtual void handleInitiateAck(const IEvent& initiateAckEvent);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

handleTerminate

Handles IDDETopicServer::endConversation requests from server applications.

See endConversation (p. 360) for information about that member.

virtual void handleTerminate(const IEvent& terminateEvent);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Event Processing

Event-processing members are called to provide you with information when events are received from the DDE server application. To obtain the information provided by any of these virtual members, you must derive from this class and override the members. All of the members, except for the data member which is pure virtual, have default implementations so that you only need to override the members that provide information you want to process.

If you specify true for the useEventThread parameter of the IDDEClientConversation constructor, all of these members are invoked on a secondary thread.

IDDEClientConversation

acknowledged

Provides generic acknowledgment from the server to a client-initiated transaction. To get information provided by this function, provide a specialized implementation in the derived class.

A positive acknowledgment can be sent in response to an IDDEClientConversation::beginHotLink or IDDEClientConversation::endHotLink member. A negative acknowledgment can be sent in response to the same two members, as well as to an IDDEClientConversation::requestData member.

See beginHotLink (p. 314), endHotLink (p. 315), and requestData (p. 316) for information about those members.

If true is specified for the useEventThread argument of the IDDEClientConversation constructor, this member is called on a secondary thread.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
acknowledged(IDDEClientAcknowledgeEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

conversationEnded

Notifies the client that the conversation is ending or ended. The conversation end can be initiated by either the client or the server, and it can also be caused by an error condition in the IDDEClientConversation.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
conversationEnded(IDDEClientEndEvent& event);	<i>Y</i>	<i>Y</i>	<i>N</i>

data

Provides data sent from the server. To get information provided by this member, provide a specialized implementation in the derived class.

The data can be from an active hot link or as the result of the IDDEClientConversation::requestData member being called. See requestData (p. 316) for information about that member.

If the server has requested an acknowledgment, the IDDEClientConversation object uses the return value from this member to determine whether to send a positive or negative acknowledgment. If the data is for a valid item and in the correct format, return true; otherwise, return false.

If true is specified for the useEventThread argument of the IDDEClientConversation constructor, this member is called on a secondary thread.

This member is pure virtual and must be overridden.

IDDEClientConversation

virtual Boolean data(IDDEDataEvent& event) = 0;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

executeAcknowledged

Provides a positive or negative acknowledgment from the server as the result of the IDDEClientConversation::executeCommands member being called. To get information provided by this member, provide a specialized implementation in the derived class. See executeCommands (p. 315) for information about that member.

If true is specified for the useEventThread argument of the IDDEClientConversation constructor, this member is called on a secondary thread.

virtual void executeAcknowledged(IDDEAcknowledgeExecuteEvent& event);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

hotLinkInform

Notifies the client that the value of an item in the server for which the client has a hot link has changed. To get information provided by this member, provide a specialized implementation in the derived class.

If the server has requested an acknowledgment, the IDDEClientConversation object uses the return value from this member to determine whether to send a positive or negative acknowledgment. If the client has an active hot link for this item and has requested that only notifications be sent, return true; otherwise, return false. The default behavior is to return true.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

virtual Boolean hotLinkInform(IDDEClientHotLinkEvent& event);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

pokeAcknowledged

Provides a positive or negative acknowledgment from the server as the result of the IDDEClientConversation::pokeData member being called. To get information provided by this member, provide a specialized implementation in the derived class. If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

See pokeData (p. 316) and IDDETopicServer (p. 358) for information about that member and class.

IDDEClientConversation

```
virtual void
pokeAcknowledged( IDDEAcknowledgePokeEvent& event );
```

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

Miscellaneous Implementation

The IDDEClientConversation object uses these members to provide portions of its implementation. Typically you do not need to override these members.

endAllHotLinks

Called by the IDDEClientConversation::endHotLinks function to update hot link instance data when one or more hot links are ending. The first version of this overloaded member is called when all hot links on an item are ending. The other version, which accepts no arguments, is used when all hot links are ending.

See endHotLinks (p. 315) for information about that member.

```
1 virtual void
endAllHotLinks( const char* item );
```

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

Exceptions	
InvalidParameter	The item parameter has a zero length.
IAccessError	The operating system's request to post the DDE message to the server application failed.
InvalidRequest	There are no active hot links for this item, or an endHotLinks request for this item is pending.

```
2 virtual void
endAllHotLinks();
```

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

Exceptions	
IAccessError	The operating system's request to post the DDE message to the server application failed.
InvalidRequest	There are no active hot links, or an endHotLinks request for all hot links is pending.

findTransaction

Called by any of the handle members, such as IDDEClientConversation::handleAck and IDDEClientConversation::handleData, when it is suspected that the server is not sending acknowledgments when it should. If this member determines that the suspicion is correct, it simulates acknowledgments for the transactions that are waiting for these acknowledgments.

IDDEClientConversation

```
virtual Boolean
  findTransaction( const IEvent& event,
                  Boolean removeMatch = false );
```

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

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEClientEndEvent

Derivation IBase
 IVBase
 IEvent
 IDDEEndEvent
 IDDEClientEndEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	323	topic	324
	application	324	~IDDEClientEndEvent	324

The IDDEClientEndEvent class provides event information to a client application when a conversation is ended or ending.

An object of this class is created when an IDDEClientConversation object needs to notify the application that a conversation is ending or ended. You do not construct objects of this class. See IDDEClientConversation (p. 309) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDEClientConversation objects.

IDDEClientEndEvent

IDDEClientEndEvent(const IEvent& ddeEvent,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
Source endSource,	<i>Y</i>	<i>Y</i>	<i>N</i>
IString application,			
IString topic);			

IDDEClientEndEvent

~IDDEClientEndEvent

~IDDEClientEndEvent();

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

Event Information

Use these members to query information about the conversation that is ending.

application Returns the name of the server application for the conversation that is ending or ended.

IString
application() const;

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

topic Returns the name of the topic supported by the conversation that is ending or ended.

IString
topic() const;

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

Inherited Public Functions

IDDEndEvent		
sourceOfEnd		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

IDDEClientEndEvent

IBase		
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEClientHotLinkEvent

IDDEClientHotLinkEvent



Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	326	isDataRequested	327
	isAckRequested	327	~IDDEClientHotLinkEvent	327

The IDDEClientHotLinkEvent class provides information to a client application when a change notification for a hot link item is sent by a server application.


An object of this class is created when an IDDEClientConversation object needs to pass a hot link notification to the client application. An object of this class for each active hot link is also kept in a set by IDDEClientConversation objects to keep track of active hot links. You do not construct objects of this class. See IDDEClientConversation (p. 309) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are usually created and deleted for you by IDDEClientConversation objects.

IDDEClientHotLinkEvent

 IDDEClientHotLinkEvent(const IEvent& ddeEvent);

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

IDDEClientHotLinkEvent

2	IDDEClientHotLinkEvent(const IDDEClientHotLinkEvent& ddeEvent);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

~IDDEClientHotLinkEvent

~IDDEClientHotLinkEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to query information about the hot link event.

isAckRequested

Returns true if the server has requested an acknowledgment of receipt of the notification. The IDDEClientConversation object automatically sends the acknowledgment when it is requested.

See IDDEClientConversation (p. 309) for information about that class.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isAckRequested() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isDataRequested

Returns true if the hot link is a data hot link. False is returned if it is a notification hot link. This function is only pertinent for objects of this class used to keep track of active hot links (contained in an IDDEClientHotLinkSet object). All other objects of this class are created for notifications sent by non-data hot links.

See IDDEClientHotLinkSet (p. 329) for information about that class.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isDataRequested() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEClientHotLinkEvent

IDDEEvent		
format	item	

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEClientHotLinkSet

Derivation ISet
 IDDEClientHotLinkSet

Inherited By None.

Header File iddecset.hpp

Members	Member	Page
	Constructor	329
	~IDDEClientHotLinkSet	330

The IDDEClientHotLinkSet class is a set that was created using the ISet template class. The set contains pointers to IDDEClientHotLinkEvent objects. See IDDEClientHotLinkEvent (p. 326) for information about that class.

An IDDEClientConversation object creates objects of this class to keep track of the active hot links for the conversation. An application creates an object of this class to call the IDDEClientConversation::hotLinks member to get the current set of active hot links. If you remove individual elements of the set, delete the IDDEClientHotLinkEvent to which each element points or the memory is not recovered. See IDDEClientConversation (p. 309) and hotLinks (p. 314) for information about that class and member.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDDEClientHotLinkSet

```
IDDEClientHotLinkSet();
```

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

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

IDDEClientHotLinkSet

~IDDEClientHotLinkSet

This member removes all the elements from the set and deletes them.

`~IDDEClientHotLinkSet();`

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



IDDEDataEvent

Derivation

```

IBase
IVBase
IEvent
IDDEEvent
IDDESetAcknowledgeInfoEvent
IDDEDataEvent
  
```

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	331	isDataFromHotLink	332
	data	332	~IDDEDataEvent	332
	isAckRequested	332		

The IDDEDataEvent class provides event information and data to a client application.

An object of this class is created when an IDDEClientConversation object needs to pass data sent from a server to the client application. This can result from a requestData request or from an active hot link. You do not construct objects of this class. See IDDEClientConversation (p. 309) and requestData (p. 316) for information about that class and member.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are usually created and deleted for you by IDDEClientConversation objects.

IDDEDataEvent

```
IDDEDataEvent( const IEvent& ddeEvent );
```

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

IDDEDataEvent

~IDDEDataEvent

~IDDEDataEvent();

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

Event Information

Use these members to query the data and other related information from the event.

data Returns the data provided by the server.

Note: The IDDEDataEvent::data member's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and member.

IString
data() const;

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

isAckRequested

Returns true if the server has requested an acknowledgment of receipt of the data. The IDDEClientConversation object automatically sends the acknowledgment when it is requested.

See IDDEClientConversation (p. 309) for information about that class.

Boolean
isAckRequested() const;

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

isDataFromHotLink

Returns true if the data is being sent as the result of an active hot link. Returns false if the data is sent in response to an IDDEClientConversation::requestData call.

See requestData (p. 316) for information about that member.

Boolean
isDataFromHotLink() const;

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

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEEvent		
format	item	

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

IVBase		
asDebugInfo	asString	

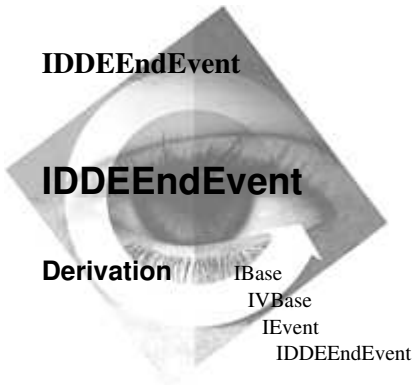
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By IDDEClientEndEvent

Header File iddeevt.hpp

Members	Member	Page
	Constructor	334
	sourceOfEnd	335
	~IDDEEndEvent	334

The IDDEEndEvent class provides event information to a server application when a conversation is ended or ending.

An object of this class is created when an IDDETopicServer object needs to notify the application that a conversation is ending or ended. You do not construct objects of this class. See IDDETopicServer (p. 358) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are usually created and deleted for you by IDDETopicServer objects.

IDDEEndEvent

IDDEEndEvent(const IEvent& ddeEvent, Source endSource);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

~IDDEEndEvent

IDDEndEvent

```
~IDDEndEvent();
```

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

Event Information

Use these members and enumerations to determine the initiator or cause of the conversation termination.

sourceOfEnd Returns one of the values of the IDDEndEvent::Source enumeration indicating the source of the ending of the conversation.

See Source (p. 336) for information about that enumeration.

```
Source  
sourceOfEnd() const;
```

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

Inherited Public Functions

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IDDEndEvent

Source

```
Source {  
    client,  
    server,  
    error  
};
```

The following enumerations are used to define the possible sources of an end to a conversation:

client

Indicates that the client initiated the conversation's end.

server

Indicates that the server initiated the conversation's end.

error

Indicates that an error in a routine entered from the dispatcher would cause an exception to be thrown and the application would not have an opportunity to catch the exception. Because this situation could cause this application and the application being conversed with to abnormally end, this application is called back with an IDDEndEvent object. The conversation is also terminated at this point.



IDDEEvent

Derivation

```

IBase
IVBase
IEvent
IDDEEvent
  
```

Inherited By

```

IDDEAcknowledgeEvent
IDDESetAcknowledgeInfoEvent
  
```

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	337	setBuffer	339
	buffer	339	setStatus	339
	format	338	status	339
	item	338	~IDDEEvent	338

IDDEEvent is the base class for most Dynamic Data Exchange (DDE) event classes.

An object of this class, or one of its derived classes, is created when an IDDEClientConversation object or IDDETopicServer object needs to pass information to the client or server application. You do not construct objects of this class. See IDDEClientConversation (p. 309) and IDDETopicServer (p. 358) for information about those classes.

Public Functions

Constructors


You can construct and destruct objects of this class, although normally you will never need to do so. These events are usually created and deleted for you by IDDETopicServer or IDDEClientConversation objects.

IDDEEvent

1 IDDEEvent(const IEvent& ddeEvent);

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

IDDEEvent

 IDDEEvent(const IDDEEvent& ddeEvent);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IDDEEvent

~IDDEEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Event Information

Use these members to query information about the item and format of the data that the event is dealing with.

format Returns the name of the format.

IString format() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

item Returns the name of the item.

IString item() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

IDDEEvent

IBase		
asString	messageText	version

Protected Functions

Implementation

Derived classes use these members to set and obtain the attributes of this class.

buffer

Returns the data buffer.

Note: The IDDEEvent::buffer member's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and member.

IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
buffer() const;	Y	Y	N

setBuffer

Sets the data into the data buffer.

IDDEEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setBuffer(IString dataBuffer);	Y	Y	N

setStatus

Sets the status field of the event.

IDDEEvent&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setStatus(unsigned short status);	Y	Y	N

status

Returns the status of the event.

unsigned short	<u>Win</u>	<u>PM</u>	<u>Motif</u>
status() const;	Y	Y	N

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEExecuteEvent

IDDEExecuteEvent

Derivation

IBase
IVBase
IEvent
IDDEEvent
IDDESetAcknowledgeInfoEvent
IDDEExecuteEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page
	Constructor	340
	commands	341
	~IDDEExecuteEvent	341

The IDDEExecuteEvent class provides event information to a server application when a client has asked the server to execute a command string.

An object of this class is created when an IDDETopicServer object needs to pass an executeCommands request to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) and executeCommands (p. 366) for information about that class and function.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDEExecuteEvent

```
IDDEExecuteEvent( const IEvent& ddeEvent );
```

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

IDDEExecuteEvent

~IDDEExecuteEvent

~IDDEExecuteEvent();

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

Event Information

Use these members to query the command string sent by the DDE client application.

commands Returns the command string that the client has asked the server to execute.

Note: The IDDEExecuteEvent::commands member's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and member.

IString
commands() const;

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

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEEvent		
format	item	

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

IDDEExecuteEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEPokeEvent

Derivation

```

IBase
IVBase
IEvent
IDDEEvent
IDDESetAcknowledgeInfoEvent
IDDEPokeEvent
  
```

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page
	Constructor	343
	pokedData	344
	~IDDEPokeEvent	344

The IDDEPokeEvent class provides event information to a server application when a client has asked the server to set an item to a new value.

An object of this class is created when an IDDETopicServer object needs to pass a pokeData request to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) and pokeData (p. 367) for information about that class and member.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDEPokeEvent

```
IDDEPokeEvent( const IEvent& ddeEvent );
```

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

IDDEPokeEvent

~IDDEPokeEvent

~IDDEPokeEvent();

Win

PM

Motif

Y

Y

N

Event Information

Use these members to query the data that the DDE client application has sent to be poked.

pokedData

Returns the data that the client asked the server to set an item's value to.

Note:

The IDDEPokeEvent::pokedData member's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and member.

IString

pokedData() const;

Win

PM

Motif

Y

Y

N

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEEvent		
format	item	

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

IDDEPokeEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDERequestDataEvent

IDDERequestDataEvent

Derivation

- IBase
- IVBase
- IEvent
- IDDEEvent
- IDDESetAcknowledgeInfoEvent
- IDDERequestDataEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	346	setData	347
	requestAck	347	~IDDERequestDataEvent	346

The IDDERequestDataEvent class provides event information to a server application when a client has requested data.

An object of this class is created when an IDDETopicServer object needs to pass requests for data to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDERequestDataEvent

IDDERequestDataEvent(const IEvent& ddeEvent);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IDDERequestDataEvent

~IDDERequestDataEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDDERequestDataEvent

Event Information

Use these members to provide the data requested by the DDE client application and to set the related attributes of the event.

requestAck Sets the request acknowledgment flag.

IDDERequestDataEvent& requestAck();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

setData Provides the requested data so it can be sent to the client. This member is overloaded so that either buffers of data or character strings can be easily sent.

1	IDDERequestDataEvent& setData(const void* dataBuffer, unsigned long dataLength);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------	--	-----------------	----------------	-------------------

2	IDDERequestDataEvent& setData(const char* dataString);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
----------	---	-----------------	----------------	-------------------

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEEvent		
format	item	

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

IVBase		
asDebugInfo	asString	

IDDERequestDataEvent

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEServerAcknowledgeEvent

Derivation IBase
 IVBase
 IEvent
 IDDEEvent
 IDDEAcknowledgeEvent
 IDDEServerAcknowledgeEvent

Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	349	isAckToHotLinkUpdate	350
	data	350	~IDDEServerAcknowledgeEvent	349

The IDDEServerAcknowledgeEvent class provides event information to a server application regarding an acknowledgment from a client.

An object of this class is created when an IDDETopicServer object needs to pass information about an acknowledgment to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) for information about that class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDEServerAcknowledgeEvent

IDDEServerAcknowledgeEvent(const IEvent& ddeEvent);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IDDEServerAcknowledgeEvent

~IDDEServerAcknowledgeEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IDDEServerAcknowledgeEvent

Event Information

Use these members to query information about an acknowledgment received from a DDE client application.

data Returns the data the server provided to the client.

Note: The IDDEServerAcknowledgeEvent:: data member's IString is constructed from a void* and length. For character strings, if the terminating NULL is included in the length, it is in the buffer. Before performing string operations, such as concatenation, on this string, use IString::stripTrailing to strip any trailing NULL character.

See IString (Vol. I) and stripTrailing (Vol. I) for information about that class and member.

IString
data() const;

Win

PM

Motif

Y

Y

N

isAckToHotLinkUpdate

Returns true if the acknowledgment is in response to IDDETopicServer::hotLinkUpdate being called. It returns false if it is in response to data being provided for an IDDETopicServer::requestData call.

See hotLinkUpdate (p. 360) and requestData (p. 367) for information about those members.

Boolean
isAckToHotLinkUpdate() const;

Win

PM

Motif

Y

Y

N

Inherited Public Functions

IDDEAcknowledgeEvent		
applicationSpecificData	isAckPositive	isApplicationBusy

IDDEEvent		
format	item	

IEvent		
controlHandle	operator =	setDispatchingHandle

IDDEServerAcknowledgeEvent

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEAcknowledgeEvent		
transactionType		

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDEServerHotLinkEvent

IDDEServerHotLinkEvent



Inherited By None.

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	352	isPacingRequested	353
	isDataRequested	353	~IDDEServerHotLinkEvent	352

The IDDEServerHotLinkEvent class provides event information to a server application when a client has asked the server to begin a hot link.

An object of this class is created when an IDDETopicServer object needs to pass a beginHotLink request to the server application. You do not construct objects of this class. See IDDETopicServer (p. 358) and beginHotLink (p. 365) for information about that class and function.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer objects.

IDDEServerHotLinkEvent

```
IDDEServerHotLinkEvent( const IEvent& ddeEvent );
```

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

~IDDEServerHotLinkEvent

IDDEServerHotLinkEvent

```
~IDDEServerHotLinkEvent();
```

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

Event Information

Use these members to query information about the hot link associated with this event.

isDataRequested

Returns true if the client application has requested a data hot link. Returns false if a notification link has been requested.

```
Boolean  
isDataRequested() const;
```

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

isPacingRequested

Returns true if the client application has requested the server to request an acknowledgment from the client whenever it sends data or a notification.

```
Boolean  
isPacingRequested() const;
```

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

Inherited Public Functions

IDDESetAcknowledgeInfoEvent		
setApplicationBusy	setApplicationSpecificData	setMessageNotUnderstood

IDDEEvent		
format	item	

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

IDDEServerHotLinkEvent

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDDESetAcknowledgeInfoEvent

Derivation

```

IBase
  IVBase
    IEvent
      IDDEEvent
        IDDESetAcknowledgeInfoEvent
  
```

Inherited By

IDDEClientHotLinkEvent	IDDEPokeEvent
IDDEDataEvent	IDDERequestDataEvent
IDDEExecuteEvent	IDDEServerHotLinkEvent

Header File iddeevt.hpp

Members	Member	Page	Member	Page
	Constructor	355	setMessageNotUnderstood	356
	setApplicationBusy	356	~IDDESetAcknowledgeInfoEvent	356
	setApplicationSpecificData	356		

The IDDESetAcknowledgeInfoEvent class is the base class for Dynamic Data Exchange (DDE) event information classes that need to set acknowledgment information.

An object of a derived class of IDDESetAcknowledgeInfoEvent is only created for events that the application can respond to with a negative acknowledgment. You do not construct objects of this class.

Public Functions

Constructors

You can construct and destruct objects of this class, although normally you will never need to do so. These events are normally created and deleted for you by IDDETopicServer or IDDEClientConversation objects.

IDDESetAcknowledgeInfoEvent

```
1 IDDESetAcknowledgeInfoEvent( const IEvent& ddeEvent );
```

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

IDDESetAcknowledgeInfoEvent

2

IDDESetAcknowledgeInfoEvent(
 const IDDESetAcknowledgeInfoEvent& ddeEvent);

Win
Y

PM
Y

Motif
N

~IDDESetAcknowledgeInfoEvent

~IDDESetAcknowledgeInfoEvent();

Win
Y

PM
Y

Motif
N

Event Information

Use these members to set the information for an acknowledgment being sent to the DDE partner application.

setApplicationBusy

Sets the application busy flag. This is used by a client or server application to indicate why a request or response to a request cannot be processed.

IDDESetAcknowledgeInfoEvent&
 setApplicationBusy();

Win
Y

PM
Y

Motif
N

setApplicationSpecificData

Sets application-specific information. This can be used when a client and server application have a predefined protocol for exchanging information.

IDDESetAcknowledgeInfoEvent&
 setApplicationSpecificData(unsigned char applicationData);

Win
Y

PM
Y

Motif
N

setMessageNotUnderstood

Sets the message-not-understood flag. This is used by a client or server application to indicate why a request or response to a request cannot be processed.

IDDESetAcknowledgeInfoEvent&
 setMessageNotUnderstood();

Win
Y

PM
Y

Motif
N

Inherited Public Functions

IDDEEvent		
format	item	

IDDESetAcknowledgeInfoEvent

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

IVBase		
asDebugInfo	asString	

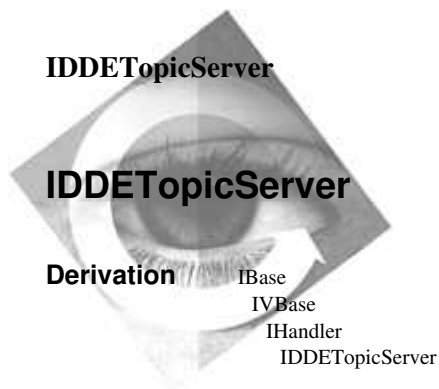
IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IDDEEvent		
buffer	setBuffer	setStatus

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File iddetsrv.hpp

Members				
Member	Page	Member	Page	
Constructor	359	handlePoke	363	
acceptConversation	365	handleRequest	364	
acknowledged	365	handleTerminate	364	
application	361	handleUnadvise	364	
beginConversation	360	hotLinkCount	361	
beginHotLink	365	hotLinkEnded	367	
conversationCount	361	hotLinkUpdate	360	
conversationEnded	366	pokeData	367	
dispatchHandlerEvent	363	removeLink	364	
endConversation	360	requestData	367	
executeCommands	366	requestHotLinkData	368	
handleAck	363	serverHandle	361	
handleAdvise	363	topic	362	
handleExecute	363	~IDDETopicServer	359	
handleInitiate	363			

The IDDETopicServer class adds Dynamic Data Exchange (DDE) server function to an application.

Create an object of this class for each topic you want to support in a DDE server application. The object manages all window, shared memory, and atom table processing.

This class uses a window to communicate; therefore, window message processing must occur. This means that ICurrentThread::processMsgs must be called. There are several ways for this to occur. Normally, this is accomplished by calling IApplication::current().run(). See processMsgs (Vol. II), current (Vol. II), and run (Vol. II) for information about those members.

Public Functions

Constructors

You can construct and destruct objects of this class.

IDDETopicServer

```
IDDETopicServer( const char* applicationName,           Win PM Motif
                  const char* supportedTopic,           Y  Y  N
                  IFrameWindow* owner = 0,
                  Boolean useEventThread = true );
```

You can construct objects of this class in the following way; The constructor accepts the following two required arguments and two optional arguments:

- The name of the application to which the topic server belongs (required).
- The name of the topic that the topic server supports (required).
- The third argument is optional but highly recommended if the application has a main frame window. IDDETopicServer creates an IFrameWindow object, which must be destructed for the application to end normally. Specifying this argument guarantees that this window is destructed when the main frame window is closed. The alternative is to ensure that all objects of IDDETopicServer are deleted before attempting to end the application. See IFrameWindow (Vol. II) for information about that class.
- The fourth argument is also optional but also highly recommended if the application must do any extensive processing or if it interacts with the user during any of the callback functions. Specifying true allows the IDDETopicServer object to create a secondary thread to process incoming events. This prevents problems with window message processing because the thread is created without a message queue. If you specify false, no secondary thread is created, and you should return promptly from all callback functions.

Exceptions	
InvalidParameter	The applicationName or supportedTopic parameter is NULL or has a zero length.
IOutOfSystemResource	The operating system's request for either an event semaphore or a queue failed. See the text of the exception for further information.

~IDDETopicServer

The destructor ends all conversations with all DDE client applications as part of its cleanup.

IDDETopicServer

```
virtual  
    ~IDDETopicServer();
```

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

Conversation Initiation and Termination

Use these members to begin and end conversations with DDE client applications.

beginConversation

This is provided along with the IDDETopicServer::serverHandle member to allow a topic server to get into conversation with a client without engaging in the normal conversation initialization. See serverHandle (p. 361) for information about that member.

This member sets the window handle associated with the client conversation. To initiate a conversation in this manner, the client and server application must have their own method for exchanging their window handles.

```
virtual IDDETopicServer&  
    beginConversation( const IWindowHandle& clientHandle );
```

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

Exceptions	
IInvalidParameter	The client window handle is not valid.
IInvalidRequest	A conversation with this client window is already active.

endConversation

Ends a conversation with a DDE client.

```
virtual IDDETopicServer&  
    endConversation( unsigned long conversationId );
```

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

Exceptions	
IInvalidRequest	There is no active conversation identified by this ID.

Hot Link Updates

Use these members to initiate the sending of hot link updates to DDE client applications.

hotLinkUpdate

Sends either data or a change notification for items whose data has changed to all clients who have an active hot link for this item.

IDDETopicServer::requestHotLinkData is called for each format that has an active hot link and requires data. The number of hot links for which data or a notification is

IDDETopicServer

sent is returned to the caller. See requestHotLinkData (p. 368) for information about that member.

If there is an outstanding acknowledgment for a hot link, the notification is not sent to the client to which the hot link belongs. When the acknowledgment is satisfied, a notification is sent to the client with the most current data if it is a data hot link.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hotLinkUpdate(const char* item);	<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidParameter	The item parameter is NULL or has a zero length.
IAccessError	The operating system's request to post the DDE message to a client application failed.
InvalidRequest	There is no active hot link for this item and format.

Server Information

Use these members to query information about the attributes of objects of this class.

application Returns the name of the server application.

IString	<u>Win</u>	<u>PM</u>	<u>Motif</u>
application() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

conversationCount

Returns the number of conversations in which the topic server is currently engaged.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
conversationCount() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

hotLinkCount

Returns the number of hot links in which the topic server is currently engaged.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hotLinkCount() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

serverHandle Returns the window handle of the topic server. This is provided along with the IDDETopicServer::beginConversation member to allow a topic server to get in conversation with a client without engaging in the normal conversation initialization. Only a client is normally allowed to initiate a conversation. See beginConversation (p. 360) for information about that member.

IDDETopicServer

```
IWindowHandle  
serverHandle() const;
```

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

topic Returns the name of the supported topic.

```
IString  
topic() const;
```

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

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members process events sent by DDE client applications. The `dispatchHandlerEvent` function is called whenever an event is sent to this object, and it calls one of the other appropriate members in this object to actually process the event.

Typically, you do not need to override any of these members. If you want to provide some additional behavior before or after the event is processed, make sure that you call the IDDETopicServer version of the member or unpredictable results can occur and the conversation may not be able to process further events from the DDE client application. You may need to override one of these members if the client application sends a DDE message not supported by the DDE protocol at that particular point in the conversation. You may need to do this because the IDDETopicServer object throws these invalid events away.

IDDETopicServer

dispatchHandlerEvent

Calls the appropriate virtual member if a DDE topic server event is found.

virtual Boolean
dispatchHandlerEvent(IEvent& event);

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

Exceptions	
IOutOfSystemResource	The operating system's request to write to a queue failed. See the text of the exception for further information.
IAccessError	This function has been called recursively, possibly violating DDE synchronization rules. Either construct this object with useEventThread=true or avoid displaying dialog boxes in the event callback functions.

handleAck

Handles acknowledgments from client applications.

virtual void
handleAck(const IEvent& ackEvent);

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

handleAdvise

Handles IDDETopicServer::beginHotLink requests from client applications. See beginHotLink (p. 365) for information about that member.

virtual void
handleAdvise(const IEvent& adviseEvent);

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

handleExecute

Handles IDDETopicServer::executeCommands requests from client applications. See executeCommands (p. 366) for information about that member.

virtual void
handleExecute(const IEvent& executeEvent);

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

handleInitiate

Handles begin requests from client applications.

virtual void
handleInitiate(const IEvent& initiateEvent);

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

handlePoke

Handles IDDETopicServer::pokeData requests from client applications. See pokeData (p. 367) for information about that member.

IDDETopicServer

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handlePoke(const IEvent& pokeEvent);	<i>Y</i>	<i>Y</i>	<i>N</i>

handleRequest

Handles IDDETopicServer::requestData requests from client applications. See requestData (p. 367) for information about that member.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleRequest(const IEvent& requestEvent);	<i>Y</i>	<i>Y</i>	<i>N</i>

handleTerminate

Handles end requests from client applications.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleTerminate(const IEvent& terminateEvent);	<i>Y</i>	<i>Y</i>	<i>N</i>

handleUnadvise

Handles IDDETopicServer::hotLinkEnded requests from client applications. See hotLinkEnded (p. 367) for information about that member.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleUnadvise(const IEvent& unadviseEvent);	<i>Y</i>	<i>Y</i>	<i>N</i>

removeLink

Called by the IDDETopicServer::handleUnadvise function to update hot link information when a client ends a hot link. See handleUnadvise (p. 364) for information about that member.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
removeLink(IString item,	<i>Y</i>	<i>Y</i>	<i>N</i>
IString format,			
unsigned long conversationId);			

Event Processing

Event-processing members provide you with information when events, typically requests, are received from a DDE client application. To obtain the information provided by any of these virtual members and to support the requested transactions, derive from this class and override the members. All of the members, except for the requestData member which is pure virtual, have default implementations so that you only need to override the functions that you want to support or use.

IDDETopicServer

If you specify true for the useEventThread parameter of the IDDETopicServer constructor, all of these members except for acceptConversation and, in some cases, requestHotLinkData, are invoked on a secondary thread.

acceptConversation

Notifies the server that a client is asking to begin a conversation. The topic server calls this member if the application and topic match, ignoring mismatches due to different cases. The application should return promptly from this member because the request is sent, not posted, by the client application. Therefore, this member is always called in the main thread.

Return true if the conversation is accepted. (The application can indicate it enforces case sensitivity by calling IDDEBeginEvent::setCaseSensitive. See setCaseSensitive (p. 304) for information about that member.)

If the conversation request is rejected, return false so the topic server does not accept the conversation. The default behavior is to return true.

virtual Boolean		Win	PM	Motif
acceptConversation(unsigned long conversationId,		<i>Y</i>	<i>Y</i>	<i>N</i>
IDDEBeginEvent& event);				

acknowledged

Notifies the server that a client has acknowledged the receipt of data or notification of changed hot link data. If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

Note: The IDDETopicServer::hotLinkUpdate and IDDETopicServer::endConversation members must not be called from this member. Otherwise, a deadlock can occur.

See hotLinkUpdate (p. 360) and endConversation (p. 360) for information about those members.

virtual void		Win	PM	Motif
acknowledged(unsigned long conversationId,		<i>Y</i>	<i>Y</i>	<i>N</i>
IDDEServerAcknowledgeEvent& event);				

beginHotLink Notifies the server that a client is requesting a hot link on a particular item and format. If the application supports hot links for this item and format, return true so the topic server sends the client a positive acknowledgment. The IDDETopicServer::hotLinkUpdate member is provided for sending either data or a notification when the item's value changes. See hotLinkUpdate (p. 360) for information about that member.

IDDETopicServer

If the application does not support this hot link request, it should use any appropriate members of the IDDEServerHotLinkEvent object to indicate the reason and return false. This causes the topic server to send a negative acknowledgment to the client. See IDDEServerHotLinkEvent (p. 352) for information about that class.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

The default behavior is to return false.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
beginHotLink(unsigned long conversationId, IDDEServerHotLinkEvent& event);	Y	Y	N

conversationEnded

Notifies the server that the conversation is ending or ended. The conversation end can be initiated by either the client or the server, and it can also be caused by an error condition in the IDDEClientConversation. See IDDEClientConversation (p. 309) for information about that class.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
conversationEnded(unsigned long conversationId, IDDEEndEvent& event);	Y	Y	N

executeCommands

Notifies the server that a client is requesting the application to execute a string of one or more commands. If the application supports the request and successfully executes the commands, return true so the topic server sends the client a positive acknowledgment. If the application cannot honor this request, it should use any appropriate members of the IDDEExecuteEvent object to indicate the reason and return false. This causes the topic server to send a negative acknowledgment to the client. The default behavior is to return false. See IDDEExecuteEvent (p. 340) for information about that class.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
executeCommands(unsigned long conversationId, IDDEExecuteEvent& event);	Y	Y	N

IDDETopicServer

hotLinkEnded

Notifies the server that a client has ended one or more hot links. If the format is a 0-length string, all hot links on the specified item are ended. If the item is a 0-length string, all hot links with this client are ended.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hotLinkEnded(unsigned long conversationId,	Y	Y	N
IDDEEvent& event);			

pokeData

Notifies the server that a client is requesting it to set an item to the new value provided by the client. If the application is able to honor the request, return true so the topic server sends the client a positive acknowledgment. If the application is unable to honor the request, it should use any appropriate members of the IDDEPokeEvent object to indicate the reason and return false. This causes the topic server to send a negative acknowledgment to the client. The default behavior is to return false. See IDDEPokeEvent (p. 343) for information about that class.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pokeData(unsigned long conversationId,	Y	Y	N
IDDEPokeEvent& event);			

requestData

Notifies the server that a client is requesting data for an item in a specified format. If the request is for an item and format that the application supports, it should provide the data using the IDDERequestDataEvent::setData member and return true. The application can also request an acknowledgment from the client when it has received the data using the IDDERequestDataEvent::requestAck function. See setData (p. 347) and requestAck (p. 347) for information about those members.

If the application cannot provide the data, it can indicate one of several reasons using the appropriate members of IDDERequestDataEvent and return false. This causes the topic server to send the client a negative acknowledgment. This function is pure virtual and must be overridden. See IDDERequestDataEvent (p. 346) for information about that class.

If true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread.

IDDETopicServer

```
virtual Boolean
  requestData( unsigned long conversationId,
               IDDERequestDataEvent& event ) = 0;
```

Win
Y

PM
Y

Motif
N

requestHotLinkData

Informs the server that data is required for an item in a specified format. This member is called once for each format that has an active hot link that requires data when the server calls the IDDETopicServer::hotLinkUpdate member for an item. The server calls the hotLinkUpdate member when an item with active hot links changes. See hotLinkUpdate (p. 360) for information about that member.

There is no need for this member to call the IDDERequestDataEvent::requestAck member because the topic server does this automatically for all hot links that have pacing active. (The server can request an acknowledgment even if pacing is not active.) For hot links that have pacing active and an outstanding acknowledgment, the update is not sent. See requestAck (p. 347) for information about that member.

When the acknowledgment is received, the topic server requests the latest update if the data has changed while the acknowledgment was outstanding. In these cases, if true is specified for the useEventThread argument of the IDDETopicServer constructor, this member is called on a secondary thread; otherwise, it is called in the main thread.

Note: The IDDETopicServer::hotLinkUpdate and IDDETopicServer::endConversation members must not be called from this member. Otherwise, a deadlock can occur. See hotLinkUpdate (p. 360) and endConversation (p. 360) for information about those members.

```
virtual void
  requestHotLinkData( IDDERequestDataEvent& event );
```

Win
Y

PM
Y

Motif
N

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IDrawingCanvas

Derivation

- IBase
- IVBase
- INotifier
- IWindow
- IControl
- ICanvas
- IDrawingCanvas

Inherited By None.

Header File idrawcv.hpp

Members	Member	Page	Member	Page
	Constructor	370	graphicList	372
	calcMinimumSize	374	setDefaultStyle	373
	classDefaultStyle	374	setGraphicContext	371
	convertToGUIStyle	372	setGraphicList	372
	defaultStyle	372	useDefaultPaintHandler	375
	graphicContext	371	~IDrawingCanvas	371

The IDrawingCanvas class is a control class that provides support for drawing graphic objects. The drawing canvas contains an IGList to which you can add graphic objects. By default, a paint handler is attached to the drawing canvas. When paint events occur, the paint handler sets the clip region to the drawing canvas' update region and iterates through the IGList, redrawing the graphic objects necessary to update the window. If you do not add graphic objects to the IGList, the drawing canvas simply paints the window using the current background color.

If you want to use your own paint handler instead of the drawing canvas' default paint handler, construct the IDrawingCanvas object without the useDefaultPaintHandler style. The default paint handler will not be attached to the drawing canvas.

You have the option of setting the graphic context that the drawing canvas will use when drawing the graphic objects contained in the IGList object. By changing graphic contexts, you can change the appearance of the graphic objects or the device on which they are rendered.

To add graphic objects to a drawing canvas, create an IGList object and call setGraphicList. The drawing canvas iterates through the drawing canvas for its

IDrawingCanvas

IGList. You can easily change the objects drawn in the drawing canvas by changing the IGList object.

Public Functions

Constructors

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

IDrawingCanvas

```
IDrawingCanvas( unsigned long windowIdentifier,           Win PM Motif
                  IWindow* parent,                        Y   Y   N
                  IWindow* owner,
                  const IRectangle& initial = IRectangle ( ),
                  const Style& style = defaultStyle ( ) );
```

windowIdentifier

The window identifier of the drawing canvas you are constructing.

We recommend that you do the following:

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

Presentation Manager Notes: Do not use FID_xxx values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

parent

The parent window of the drawing canvas you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.

owner

The owner window of the drawing canvas you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.

Motif Notes: The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

IDrawingCanvas

<i>initial</i>	The initial position and size of the drawing canvas you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.
<i>style</i>	The window's characteristics. This value can be a combination of IDrawingCanvas::Style (p. 374), ICanvas::Style (p. 35) and IWindow::Style (Vol. II) objects. Optional.

~IDrawingCanvas

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

Graphic Context

Use these members to set and query the graphic context, which is used to paint the window when using the style IDrawingCanvas::useDefaultPaintHandler (p. 375). If you do not set a graphic context, the default paint handler uses the graphic context that is provided from the paint event. See the class IGraphicContext (Vol. IV) for more details.

graphicContext

Returns a pointer to the graphic context that the drawing canvas uses for paint events. By default, the drawing canvas uses the graphic context that is provided from the paint event.

virtual IGraphicContext*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
graphicContext() const;	Y	Y	N

setGraphicContext

Sets the graphic context that the drawing canvas uses when painting the window.

virtual IDrawingCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setGraphicContext(IGraphicContext* graphicContext);	Y	Y	N

Graphic List

The graphic list is an ordered collection of graphic objects that is drawn when the drawing canvas receives a paint event.

Use these members to set and query the graphic list object, which is drawn when the drawing canvas receives a paint event. The drawing canvas does not delete the graphic list when the drawing canvas is destroyed.

IDrawingCanvas

graphicList Returns a pointer to the IGList (Vol. IV) object set for the drawing canvas.

<pre>virtual IGList* graphicList() const;</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setGraphicList

Sets the IGList (Vol. IV) object that the drawing canvas draws when paint events occur.

<pre>virtual IDrawingCanvas& setGraphicList(IGList* list);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an IDrawingCanvas object, you can use IDrawingCanvas, ICanvas, and IWindow member functions to query and change individual styles.

convertToGUIStyle

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

<pre>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultStyle Returns the current default style, which is the same as classDefaultStyle (p. 374) unless you have changed the style using setDefaultStyle (p. 373).

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

IDrawingCanvas

setDefaultStyle

Sets the default style for all subsequent drawing canvases.

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

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

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Canvas Support

These members override the inherited canvas-protected functions to provide layout support.

IDrawingCanvas

calcMinimumSize

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

virtual ISize
calcMinimumSize() const;

Win
Y

PM
Y

Motif
N

Inherited Protected Functions

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an IDrawingCanvas object, you can use IDrawingCanvas, ICanvas, and IWindow member functions to query and change individual styles.

classDefaultStyle

Specifies the original default style for this class, which is
IDrawingCanvas::useDefaultPaintHandler | IWindow::visible | IWindow::clipSiblings.

static const Style
classDefaultStyle;

Win
Y

PM
Y

Motif
N

IDrawingCanvas

useDefaultPainter

If this style is set, a default paint handler attaches to the drawing canvas. If this style is not set, you must handle window painting.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
useDefaultPainter;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IDrawingCanvas contains the following nested classes:

IDrawingCanvas::Style (see page 376)



IDrawingCanvas::Style

IDrawingCanvas::Style

Derivation IDrawingCanvas
IBase
IBitFlag
IDrawingCanvas::Style

Inherited By None.

Header File idrawcv.hpp

The nested class IDrawingCanvas::Style provides a set of valid styles for the IDrawingCanvas::defaultStyle (p. 372) and IDrawingCanvas::setDefaultStyle (p. 373) functions, and for the constructor of the IDrawingCanvas (p. 369) class. You can use these styles with the styles defined by the following nested classes:

- ICanvas::Style (p. 35)
- IWindow::Style (Vol. II)

Once you have constructed an IDrawingCanvas object, you can use IDrawingCanvas, ICanvas, and IWindow member functions to query and change individual styles.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File ifiledlg.hpp

Members				
Member	Page	Member	Page	
Constructor	379	noStyle	387	
applyButton	386	ok	386	
buttonPressedId	382	preload	387	
cancel	386	pressedOK	382	
classDefaultStyle	386	registerCallbacks	385	
convertToGUIStyle	383	returnValue	382	
defaultStyle	383	saveAsEAType	381	
fileName	381	selectableListbox	388	
filter	386	selectedFileCount	381	
helpButton	387	setDefaultStyle	383	
id	382	setId	383	
includeEAS	387	show	384	
isModeless	382	unregisterCallbacks	385	
modeless	387	~IFileDialog	381	
multiSelection	387			

The IFileDialog class displays 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 (Vol. II)
- IFileDialogHandler (p. 400)
- IKeyboardHandler (Vol. II)
- IMouseHandler (Vol. II)



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.

IFileDialog

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.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IFileDialog objects.

IFileDialog

❏	IFileDialog(IWindow* parent,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	IWindow* owner,	Y	Y	Y
	IHandler* handler,			
	const Style& style = defaultStyle (),			
	const Settings& settings = Settings ());			

You can construct an object of this class by using this constructor, which accepts a parent, owner, handler, style, and settings as parameters.

To add a IFileDialogHandler (p. 400) 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 this constructor are the following:

<i>parent</i>	The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.
<i>owner</i>	The owner window.
<i>handler</i>	A pointer to an IHandler (Vol. II) object.
<i>style</i>	Use the styles provided by IFileDialog (p. 386) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional.
<i>settings</i>	Use the settings provided by IFileDialog::Settings (p. 390) to specify the setting. Optional.

FileDialog

2	<code>FileDialog(IWindow* parent, IWindow* owner, const Style& style = defaultStyle ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

You can construct an object of this class by using this constructor, which accepts a parent, owner, and style as parameters.

The parameters for this constructor are the following:

<i>parent</i>	The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.
<i>owner</i>	The owner window.
<i>style</i>	Use the styles provided by IFileDialog (p. 386) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional.

3	<code>FileDialog(IWindow* parent, IWindow* owner, const Settings& settings, const Style& style = defaultStyle ());</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

You can construct an object of this class by using this constructor, which accepts a parent, owner, settings, and style as parameters.

The parameters for this constructor are the following:

<i>parent</i>	The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.
<i>owner</i>	The owner window.
<i>settings</i>	Use the settings provided by IFileDialog::Settings (p. 390) to specify the setting.
<i>style</i>	Use the styles provided by IFileDialog (p. 386) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional.

4	<code>FileDialog(IWindow* parent, IWindow* owner, const Style& style, const Settings& settings);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

You can construct an object of this class by using this constructor, which accepts a parent, owner, style, and settings as parameters.

The parameters for this constructor are the following:

IFileDialog

<i>parent</i>	The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.
<i>owner</i>	The owner window.
<i>style</i>	Use the styles provided by IFileDialog (p. 386) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style.
<i>settings</i>	Use the settings provided by IFileDialog::Settings (p. 390) to specify the setting.

~IFileDialog

virtual	Win	PM	Motif
~IFileDialog();	<i>Y</i>	<i>Y</i>	<i>Y</i>

Getting Information about the Chosen File

Use these members to get information about the file that was selected in the file dialog.

fileName 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 (p. 387) for your IFileDialog object, set *fileNumber* to selectedFileCount (p. 381) minus 1 to ensure the correct number of files is returned.

IString	Win	PM	Motif
fileName(unsigned fileNumber = 0) const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

saveAsEAType

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.

IString	Win	PM	Motif
saveAsEAType() const;	<i>N</i>	<i>Y</i>	<i>Y</i>



This function always returns an empty string.

selectedFileCount

Returns the number of files selected by the user.

FileDialog

Note: If you create the dialog with the style multiSelection (p. 387), this number can be more than 1.

unsigned selectedFileCount() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Getting Information about the Dialog

Use these members to get specific information about the dialog.

buttonPressedId

Returns the ID of the push button used to close the dialog.

unsigned long buttonPressedId() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

id

Returns the window identifier of the window.

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

isModeless

If this is a modeless dialog, true is returned.

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

pressedOK

If the user ends the dialog by selecting the **OK** push button, true is returned.

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

returnValue

If an error occurs, the return code is returned.

long returnValue() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
------------------------------	-----------------	----------------	-------------------



The return value is an FDS_ERR system return code. When a dialog fails, this value tells the application the reason for the failure.



This function always returns 0.

Setting Information about the Dialog

Use these members to set specific information about the dialog.

setId Sets the window identifier of the window.

<pre>virtual IFileDialog& setId(unsigned long newIdentifier);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>N</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>N</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>N</i>	<i>N</i>					

Styles

These style members provide a set of valid file dialog styles for the IFileDialog::setDefaultStyle function and for the constructors of the IFileDialog (p. 378) class.

convertToGUIStyle

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

<pre>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 386) unless you changed the style using setDefaultStyle (p. 383).

<pre>static Style defaultStyle();</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setDefaultStyle

Sets the default style for all subsequent file dialogs.

style Dialog style. Use the styles provided by IFileDialog (p. 386) to specify the default style.

<pre>static void setDefaultStyle(const Style& style);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Window Display

Use the window display members to manage the visibility of the file dialog.

IFileDialog

show Shows or hides the file dialog. If the flag is true, the file dialog is shown. If the flag is false the file dialog is hidden.

```
virtual IFileDialog&  
    show( Boolean showWindow = true );
```

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

Inherited Public Functions

IFrameWindow		
addExtension	isAnExtension	setBorderSize
addToWindowList	isFlashing	setBorderWidth
backgroundColor	isMaximized	setClient
beginFlashing	isMinimized	setDefaultOrdering
borderHeight	isModal	setDefaultStyle
borderSize	matchForMnemonic	setDestroyOnClose
borderWidth	maximize	setExtensionSize
client	maximizeRect	setIcon
clientHandle	minimize	setLayoutDistorted
clientRectFor	minimizeRect	setMousePointer
close	mousePointer	setRestoreRect
convertToGUIStyle	moveSizeToClient	setResult
defaultOrdering	nextShellRect	setToolBarList
defaultPushButton	notifyOwner	shareParentDBCSStatus
defaultStyle	removeExtension	show
disabledBackgroundColor	removeFromWindowList	showModally
dismiss	resetBackgroundColor	start
enableNotification	resetDisabledBackgroundColor	toolbarList
endFlashing	restore	topHandle
frameRectFor	restoreRect	update
handleFor	result	useExtensionMinimumSize
icon	setBorderHeight	usesDialogBackground

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IFileDialog

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Overrides

These members override the base class members used to implement this class.

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.

virtual void	Win	PM	Motif
registerCallbacks();	<i>N</i>	<i>N</i>	<i>Y</i>

unregisterCallbacks

Removes callbacks and X event handlers from this IFileDialog that were added by registerCallbacks (p. 385).

virtual void	Win	PM	Motif
unregisterCallbacks();	<i>N</i>	<i>N</i>	<i>Y</i>

Inherited Protected Functions

IFrameWindow		
addDefaultHandler	findExtensions	registerFrameClass
attachClient	initialize	removeDefaultHandler
create	isFrameWindow	setExtensions
extensions	isRelatedHandle	tryToLoadDialog
findExtension	registerCallbacks	unregisterCallbacks

IFileDialog

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Getting Information about the Dialog

Use these members to get specific information about the dialog.

cancel	Push button ID value returned when the dialog is dismissed by pressing the Cancel button.			
	<code>static const unsigned long cancel;</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
ok	Push button ID value returned when the dialog is dismissed by pressing the OK button.			
	<code>static const unsigned long ok;</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

Styles

These style members provide a set of valid file dialog styles for the IFileDialog::setDefaultStyle function and for the constructors of the IFileDialog (p. 378) class.

applyButton	Adds an Apply push button to the dialog. This is useful in a modeless dialog.			
	<code>static const Style applyButton;</code>	<u>Win</u> I	<u>PM</u> Y	<u>Motif</u> Y
classDefaultStyle	Specifies the original default style for this class, which is noStyle. This style results in a modal open dialog.			
	<code>static const Style classDefaultStyle;</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

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.
---------------	--

IFileDialog

	<pre>static const Style filter;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>I</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>I</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>I</i>	<i>Y</i>	<i>Y</i>						
helpButton	Adds a Help push button to the dialog. This push button sends a help message to the owner of the dialog.							
	<pre>static const Style helpButton;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						
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.							
	<pre>static const Style includeEAS;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>I</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>I</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>I</i>	<i>Y</i>	<i>Y</i>						
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 (p. 403).							
	<pre>static const Style modeless;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						
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.							
	<pre>static const Style multiSelection;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						
noStyle	Disables all of the valid styles. Use this style only when you are not using any other styles.							
	<pre>static const Style noStyle;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						
preload	Specifies that the dialog do the following: <ul style="list-style-type: none">• 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.							

IFileDialog

Note: This is not done by default because of potential delays in retrieving this information, which can be caused by network delays (for network drives) or hardware timeouts due to unavailable media.

static const Style preload;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--------------------------------	-----------------	----------------	-------------------

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.

static const Style selectableListbox;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Inherited Public Data

IFrameWindow		
activateId	closeId	deactivateId

IWindow		
activeColorId	disabledForegroundColorId	hiliteForegroundColorId
backgroundColorId	enableId	inactiveColorId
borderColorId	focusId	positionId
commandId	fontId	shadowColorId
deleteId	foregroundColorId	sizeId
disabledBackgroundColorId	hiliteBackgroundColorId	systemCommandId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IFileDialog contains the following nested classes:

IFileDialog::Settings (see page 390)

IFileDialog::Style (see page 396)

IFileDialog::Settings

IFileDialog::Settings

Derivation



```
graph BT; IBase --> IFileDialogSettings[IFileDialog::Settings]
```

Inherited By None.

Header File ifiledlg.hpp

Members		Member	Page	Member	Page
		Constructor	391	setFileName	394
		addDrive	393	setInitialDrive	394
		addFileType	393	setInitialFileType	394
		fileName	393	setOKButtonText	392
		initialDrive	393	setOpenDialog	394
		initialFileType	393	setPosition	392
		isDialogTemplateSet	391	setSaveAsDialog	395
		isOpenDialog	394	Settings	391
		isPositionSet	391	setTitle	392
		okButtonText	391	title	393
		position	391	~Settings	391
		setDialogTemplate	392		

The IFileDialog::Settings class contains input for an IFileDialog (p. 378) 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

PM

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

Public Functions

Constructors

You can construct and destruct objects of this class.

Settings Provides the default constructor, which accepts no parameters.

Settings();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~Settings

~Settings();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Dialog Customization

Use these members to customize the file dialog presented to the user. The file dialog includes the dialog's title, **OK** push button's text, dialog position, and the dialog template to use.

isDialogTemplateSet

Returns true if the dialog template was set.

Boolean isDialogTemplateSet() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

isPositionSet Returns true if the dialog position was set.

Boolean isPositionSet() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

okButtonText

Returns the text of the **OK** push button.

IStrString okButtonText() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

position Returns the initial placement of the dialog within the parent window.

IFileDialog::Settings

IPoint position() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
-----------------------------	-----------------	----------------	-------------------

setDialogTemplate

Sets a dialog template resource to use in place of the OS/2-supplied default file dialog.

resId The resource ID of the dialog template to use.

Settings& setDialogTemplate(const IResourceId& resId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

setOKButtonText

Sets the text that appears on the **OK** push button. The parameters are one of the following:

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.

1 Settings& setOKButtonText(const char* newText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

2 Settings& setOKButtonText(const IResourceId& resId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

setPosition Sets the initial placement of the dialog within the parent window.

position The initial position for the dialog.

Settings& setPosition(const IPoint& position);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

setTitle Sets the dialog's title. The parameters are one of 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.

1 Settings& setTitle(const char* newTitle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

IFileDialog::Settings

2	Settings& setTitle(const IResourceId& resId);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	--	-----------------	----------------	-------------------

title Returns the dialog's title.

IString title() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---------------------------	-----------------	----------------	-------------------

Setting Information about the Initial Dialog

Use these members to set and query the initial information that appears in the dialog.

addDrive 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.

Settings& addDrive(const char* drive);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

addFileType Adds the specified type to the drop-down list box of extended-attribute types.

fileType The extended-attribute type to add.

Settings& addFileType(const char* fileType);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

fileName Returns the default file name selection.

IString fileName() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
------------------------------	-----------------	----------------	-------------------

initialDrive Returns the drive for which initial information is displayed.

IString initialDrive() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------------------------------	-----------------	----------------	-------------------

initialFileType

Returns the file type that is used to filter the initial list of files.

IString initialFileType() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
-------------------------------------	-----------------	----------------	-------------------

IFileDialog::Settings

isOpenDialog Returns true if dialog is an open dialog.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isOpenDialog() const;	Y	Y	Y

setFileName Sets a default file name selection.

fileName The file name to use as the default. If you want to specify global (or 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.

Settings&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setFileName(const char* fileName);	Y	Y	Y

setInitialDrive

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 (p. 393).

drive The initial drive to display.

Settings&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setInitialDrive(const char* drive);	Y	Y	I

setInitialFileType

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 (p. 393).

fileType The initial extended attribute.

Settings&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setInitialFileType(const char* fileType);	Y	Y	I

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.

IFileDialog::Settings

```
Settings&  
    setOpenDialog();
```

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

setSaveAsDialog

Creates a Save As dialog. An Open File dialog is the default. By default, an Open File dialog has the list of files enabled for selection, while a Save As dialog has this list disabled. You can use the style IFileDialog::selectableListbox (p. 388) to force the list of files to be selectable on a Save As dialog.

```
Settings&  
    setSaveAsDialog();
```

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

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFileDialog::Style

IFileDialog::Style

Derivation IBase
IBitFlag
IFileDialog::Style

Inherited By None.

Header File ifiledlg.hpp

The nested class IFileDialog::Style provides a set of valid styles for the IFileDialog (p. 378) class.



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

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File ifilehdr.hpp

Members		Member	Page	Member	Page
		Constructor	398	fileName	399
		fileLength	399	~IFileDialogEvent	398

The IFileDialogEvent class encapsulates events used in an IFileDialogHandler (p. 400) object. It provides the name and length of the file that the event pertains to.

Public Functions

Constructors

You can construct and destruct objects of this class.

IFileDialogEvent

Although you can construct objects of this class, typically IFileDialogHandler::dispatchHandlerEvent (p. 402) creates objects of this class from an object of the class IEvent (Vol. II).

IFileDialogEvent(IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

~IFileDialogEvent

virtual ~IFileDialogEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

IFileDialogEvent

File Name Accessors

Use these members to retrieve information about the event.

fileLength Returns the length of the file name the event pertains to.

unsigned long fileLength() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--------------------------------------	-----------------	----------------	-------------------

fileName Returns the file name the event pertains to.

NSString fileName() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
-------------------------------	-----------------	----------------	-------------------

Inherited Public Functions

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File ifilehdr.hpp

Members				
Member	Page	Member	Page	
Constructor	401	modelessResults	403	
dispatchHandlerEvent	402	stopHandlingEventsFor	402	
filter	403	validate	404	
filterName	403	validateName	404	
handleEventsFor	401	~IFileDialogHandler	401	
modelessFileApplied	403			

The IFileDialogHandler class handles 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 (Vol. II) 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 (p. 378) that accepts such a pointer as one of its parameters.

When the file dialog handler receives a file dialog event, it creates an IFileDialogEvent (p. 398) 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.

IFileDialogHandler

- 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 (Vol. II) to process the file dialog event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IFileDialogHandler

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

IFileDialogHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~IFileDialogHandler

virtual ~IFileDialogHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Window Attachment

Use these members to attach a handler object to and detach it from a file dialog window.

handleEventsFor

Attaches the handler to the specified file dialog passed in the argument.

virtual IFileDialogHandler& handleEventsFor(IFileDialog* fileDialog);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Exceptions	
InvalidParameter	A 0 pointer was passed instead of a valid IFileDialog*.

IFileDialogHandler

stopHandlingEventsFor

Detaches the handler from the specified file dialog passed in the argument.

```
virtual IFileDialogHandler&
    stopHandlingEventsFor( IFileDialog* fileDialog );
```

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

Exceptions	
InvalidParameter	A 0 pointer was passed instead of a valid IFileDialog*.

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Dispatch Events

These members determine if the event is one of the file dialog events. If it is, it calls the appropriate virtual function.

dispatchHandlerEvent

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

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

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

IFileDialogHandler

File Dialog Events

These members are called when a file dialog event occurs. Override these to provide your own processing.

filter Lets you determine which file names appear in the list box. Use IFileDialogEvent::fileName (p. 399) 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) (Vol. II). If you do not want the event to be processed by any other handlers, return true.

```
virtual Boolean                                     Win PM Motif  
    filter( IFileDialogEvent& event );              I   Y   Y
```

filterName 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* (p. 378) pointer.

```
virtual Boolean                                     Win PM Motif  
    filterName( const IString& fileName,           I   Y   Y  
                IFileDialog* dialog );
```

modelessFileApplied

Called when the apply button of a modeless dialog is pressed. Allows you to update an application based on the current setting of the dialog without closing the dialog.

appliedDialog

Use the IFileDialog* pointer to get additional information about the dialog.

fileName The name of the file that is currently selected in the dialog.

```
virtual Boolean                                     Win PM Motif  
    modelessFileApplied( IFileDialog* appliedDialog, Y   Y   I  
                        const IString& fileName );
```

modelessResults

Called when a modeless dialog is closed, even if the dialog is cancelled. This is done so that you can destroy the IFileDialog (p. 378), if needed. Use

IFileDialogHandler

IFileDialog::pressedOK (p. 382) 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)
```

```
virtual Boolean                                     Win PM Motif
modelessResults( IFileDialog* endingDialog );      Y   Y   Y
```

validate Called when a user selects a file and presses the Enter key or double-clicks on a file in the list box. Use IFileDialogEvent::fileName (p. 399) to get the name selected by the user. If the name is unacceptable, call IEvent::setResult(false) (Vol. II). In this case, the dialog is not ended. If you do not want the event to be processed by any other handlers, return true.

```
virtual Boolean                                     Win PM Motif
validate( IFileDialogEvent& event );                Y   Y   Y
```

validateName Override this function to supply an easier way of validating the user's selection. The supplied default function returns true, so that if you do not override it, nothing is invalidated. 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 (Vol. II) constructor.

```
virtual Boolean                                     Win PM Motif
validateName( const IString& fileName,              Y   Y   Y
              IFileDialog* dialog );
```

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IFileDialogHandler

IBase		
recoverable	unrecoverable	

IFlyOverHelpHandler

IFlyOverHelpHandler



Inherited By None.

Header File iflyhhdr.hpp

Members

Member	Page	Member	Page
Constructor	408	resourceLibrary	411
defaultText	411	setDefaultText	411
delayTime	413	setDelayTime	413
dispatchHandlerEvent	414	setFlyTextControl	410
flyHelpText	409	setFlyTextStringTableOffset	412
flyTextControl	410	setHelpText	409
flyTextStringTableOffset	412	setInitialDelayTime	413
handleEventsFor	410	setLongStringTableOffset	412
initialDelayTime	413	setLongTextControl	410
longHelpText	409	setResourceLibrary	411
longStringTableOffset	412	stopHandlingEventsFor	410
longTextControl	410	~IFlyOverHelpHandler	408
removeHelpText	409		

The IFlyOverHelpHandler class displays context-specific help for windows that the mouse is over. By attaching an IFlyOverHelpHandler to a window, you can provide context-specific help messages for the window that you attached the handler to and any of its descendent windows.

You can use the IFlyOverHelpHandler to update an IFlyText control, an ITextControl, or both. Use the IFlyText control to display brief informative messages for a window, such as the function of a push button contained in a tool bar. Use ITextControl to display longer, more informative text like that found in an information area of a frame window. Typically, you will not have a string associated with every window in your application. When a help string cannot be found for a window, you can set "missing" text you would like displayed in the ITextControl. By default, a single blank is displayed in the ITextControl to avoid the frame extension handler from hiding the ITextControl when it contains a null string.

Note: It is often convenient to use the same string table identifier for the text of a control and the text that is displayed in the IFlyText control. You can have the control text span multiple lines by having a new-line character in the

IFlyOverHelpHandler

string. While the new line is desirable for the control text, it is not desirable for the IFlyText control. For this reason, new-line characters are removed from a string before they are displayed in the IFlyText control.

The last two parameters of each constructor specify time delays expressed in milliseconds. The first delay indicates an initial delay that the mouse must remain in the same location for the first time fly-over help is displayed for a window. The second delay indicates the time the mouse must remain in the same location after the first time fly-over help has been displayed. For example, the first time fly-over help is displayed, you may want to set the initial delay to two seconds. Then, you may want to set the second delay to one-half second.

The primary method of associating context-specific help for a window to a help message is by a window identifier, as is the case for IInfoArea. The window identifier, in conjunction with an offset that is added to the window identifier, is used to load strings from a string table. You can specify different offsets into the string table for the IFlyText and the ITextControl objects, thereby allowing you to display different help messages for a window in each of the controls.

Note: To display help for a window via a string table in a resource file based on the windows identifier, you must either create the window via the User Interface Class Library or wrapper an existing window.

In addition to loading help text via a string table in a resource file, you can also dynamically associate help text to a window. This association is made using window handles. This is useful if you dynamically add controls to a canvas or push buttons to a tool bar. If you dynamically add help text to a window, this takes precedence over help that is otherwise loaded from a string table.

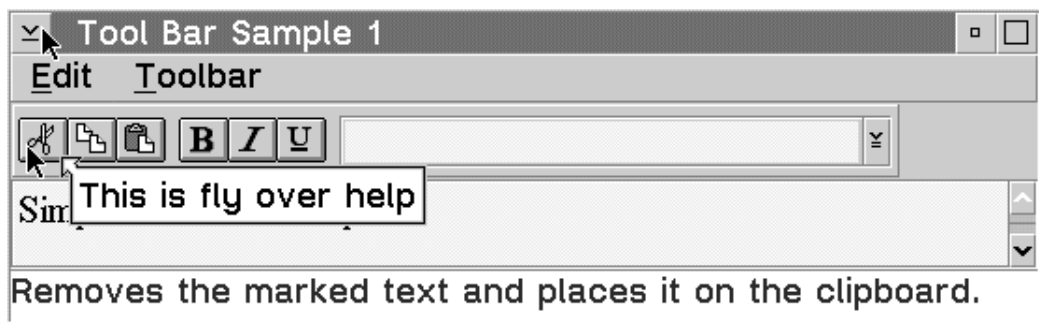


Figure 3. Fly-Over Help

IFlyOverHelpHandler

Public Functions

Constructors

You can construct and destruct objects of this class.

Note: Each constructor's last two parameters are optional. These parameters are delay times expressed in milliseconds. The first delay indicates an initial delay that the mouse must remain in the same location for the first time fly-over help is displayed for a window. The second delay indicates the time the mouse must remain in the same location after the first time fly-over help has been displayed.

IFlyOverHelpHandler

1	<code>IFlyOverHelpHandler(IFlyText* flyText, unsigned long initialDelay = 100, unsigned long delay = 100);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct objects of the IFlyOverHelpHandler class from a IFlyText object and optionally an initial delay and delay time in milliseconds. Use the IFlyText control to display brief help messages.

2	<code>IFlyOverHelpHandler(ITextControl* longText, unsigned long initialDelay = 100, unsigned long delay = 100);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct objects of the IFlyOverHelpHandler class from a ITextControl object. Typically, you display longer, more informative help messages for the window the mouse is over.

3	<code>IFlyOverHelpHandler(IFlyText* flyText, ITextControl* longText, unsigned long initialDelay = 100, unsigned long delay = 100);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Use this function to construct an object of the IFlyOverHelpHandler class from a IFlyText object and a ITextControl object.

~IFlyOverHelpHandler

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

IFlyOverHelpHandler

Dynamically Updating Help Text

These members dynamically add help text to or remove it from a window. If you add help text to a window by the setHelpText functions, this text takes precedence over text that would otherwise be loaded from a string table.

flyHelpText Returns the short help text for a window if you have dynamically added help text for the window.

virtual IString flyHelpText(const IWindowHandle& handle) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

longHelpText Returns the long help text for a window if you have dynamically added long help text for the window.

virtual IString longHelpText(const IWindowHandle& handle) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

removeHelpText

Removes help text added to a window using setHelpText.

virtual IFlyOverHelpHandler& removeHelpText(const IWindowHandle& handle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

setHelpText Sets the help text for a window by specifying a string or resource identifier.

1 virtual IFlyOverHelpHandler& setHelpText(const IWindowHandle& handle, const IString& flyText, const IString& longText = IString ());	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

2 virtual IFlyOverHelpHandler& setHelpText(const IWindowHandle& handle, const IResourceId& flyText, const IResourceId& longText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process.

IFlyOverHelpHandler

handleEventsFor

Attaches the handler to the specified window passed in the argument.

<code>virtual IFlyOverHelpHandler& handleEventsFor(IWindow* window);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

stopHandlingEventsFor

Detaches the handler from the specified window passed in the argument.

<code>virtual IFlyOverHelpHandler& stopHandlingEventsFor(IWindow* window);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Querying and Setting Text Controls

Use these members to query and set the text controls that display help text.

flyTextControl

Returns a pointer to the fly text control that is used to display fly help text.

<code>virtual IFlyText* flyTextControl() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

longTextControl

Returns a pointer to the text control that is used to display long help text.

<code>virtual ITextControl* longTextControl() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setFlyTextControl

Sets the control to use to display fly help text.

<code>virtual IFlyOverHelpHandler& setFlyTextControl(IFlyText* flyText);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setLongTextControl

Sets the control to use to display long help text.

<code>virtual IFlyOverHelpHandler& setLongTextControl(ITextControl* longText);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IFlyOverHelpHandler

Resource Library

Use these members to query and set the resource library from which the information strings are loaded.

resourceLibrary

Returns a reference to the library being used.

<code>virtual IResourceLibrary& resourceLibrary() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setResourceLibrary

Sets the resource library to be used. This can be specified as a module handle, a .dll file, or a library name.

1	<code>virtual IFlyOverHelpHandler& setResourceLibrary(const char* resDLLName);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

2	<code>virtual IFlyOverHelpHandler& setResourceLibrary(const IModuleHandle& resMod);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

Special Information Strings

Use these members to query and set the value of the information strings displayed in special circumstances. Special circumstances include when the information string is not found for the ITextControl. You can specify each special string as either an IString or the identifier for a string from the resource library.

defaultText Returns the text to display when an information string is not found for the ITextControl.

<code>virtual IString defaultText() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultText

Sets the text to display when an information string is not found for the ITextControl.

1	<code>virtual IFlyOverHelpHandler& setDefaultText(unsigned long identifier);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>N</i>

IFlyOverHelpHandler

2	<code>virtual IFlyOverHelpHandler& setDefaultText(const IString& defaultText);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

String Table Offset

Use these members to query and set a string table offset. This offset added to the window identifier for the window that the mouse is over is used to compute a value. The value is used to obtain a string from a string table. The default offset is 0.

flyTextStringTableOffset

Returns the offset currently in use for the fly text control.

<code>virtual long flyTextStringTableOffset() const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

longStringTableOffset

Returns the offset currently in use for the long text control.

<code>virtual long longStringTableOffset() const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setFlyTextStringTableOffset

Sets the offset to be used to calculate the information string identifier for the text displayed in the fly text control.

<code>virtual IFlyOverHelpHandler& setFlyTextStringTableOffset(long newOffset = 0);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

setLongStringTableOffset

Sets the offset to be used to calculate the information string identifier for the text displayed in the long text control.

<code>virtual IFlyOverHelpHandler& setLongStringTableOffset(long newOffset = 0);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Timer Delay

Use these members to query and set a timer delay. This timer delay specifies the time that the mouse must remain in the same location before help is displayed for the window under the mouse.

IFlyOverHelpHandler

delayTime Returns the minimum time, in milliseconds, the mouse must remain in the same position before help text is displayed.

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

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

initialDelayTime

Returns the minimum time, in milliseconds, the mouse must remain in the same location for the first time fly over help is displayed for a window.

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

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

setDelayTime Specifies the minimum time, in milliseconds, that the mouse must remain in the same position before help text is displayed.

```
virtual IFlyOverHelpHandler&  
    setDelayTime( unsigned long milliseconds = 100 );
```

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

setInitialDelayTime

Sets the minimum time, in milliseconds, the mouse must remain in the same location for the first time fly over help is displayed for a window.

```
virtual IFlyOverHelpHandler&  
    setInitialDelayTime( unsigned long milliseconds = 100 );
```

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

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IFlyOverHelpHandler

Protected Functions

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process.

dispatchHandlerEvent

This function intercepts events that are necessary to determine which window the mouse is over.

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

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

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IFlyText

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ITextControl
IFlyText

Inherited By

None.

Header File

iflytext.hpp

Members

Member	Page	Member	Page
Constructor	416	setRelativeWindowRect	417
hide	416	setText	416
relativeWindowRect	417	~IFlyText	416

The IFlyText class displays short help messages for the window that the mouse is currently over.

The IFlyText control displays help messages in a bordered window that is sized just large enough to contain the help text. The text is displayed using one row and is not reflowed. The default font used to display the help messages is 8-point Helvetica.

In addition to displaying the help message in a window with a border around it, IFlyText controls draw an arrow pointing outward from one of the four corners of the control. This arrow points to the window for which the help message is being displayed.

The IFlyText control positions itself relative to one of the corners of the window that the mouse is over. The objective is to display the help message such that none of the text is drawn outside of the desktop, thereby clipping the text. This also determines which corner of the IFlyText control the arrow is drawn from. There are circumstances where it is impossible to display the help message without drawing off the desktop. The order in which the IFlyText control attempts to position itself relative to the window that the mouse is over follows:

- Lower-right
- Lower-left
- Upper-left
- Upper-right



IFlyText

Public Functions

Constructors

You can construct and destruct objects of this class.

IFlyText Use this function to construct objects of the IFlyText class from a window identifier and an owning window.

<code>IFlyText(unsigned long identifier, IWindow* owner);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IFlyText

<code>virtual ~IFlyText();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Text

Use these members to set the text of the IFlyText control.

setText Sets the control window text.

1 <code>virtual IFlyText& setText(const IResourceId& text);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

2 <code>virtual IFlyText& setText(const char* text);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Window Painting

Use these members to determine the visibility of the control window.

hide Hides the control window.

<code>virtual IWindow& hide();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Window Positioning

Use these members to get and set the position that the IFlyText control will position itself relative to.

IFlyText

relativeWindowRect

Returns the rectangle of the control that the IFlyText control is positioned relative to.

```
virtual IRectangle  
    relativeWindowRect() const;
```

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

setRelativeWindowRect

Sets the rectangle of the window for which the IFlyText control will position itself relative to. This rectangle should be relative to the desktop window.

```
virtual IFlyText&  
    setRelativeWindowRect( const IRectangle& rect );
```

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

Inherited Public Functions

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

IFlyText

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Inherited Public Data

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFontDialog

Derivation

```

IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow
          IFontDialog
  
```

Inherited By None.

Header File ifontdlg.hpp

Members					
Member	Page	Member	Page	Member	Page
Constructor	420	isModeless	424		
applyButton	427	modeless	428		
bitmapOnly	427	nominalPointSize	423		
buttonPressedId	423	noStyle	428		
cancel	426	noSynthesize	428		
classDefaultStyle	427	ok	427		
convertToGUIStyle	424	pointSize	423		
defaultStyle	424	pressedOK	424		
emHeight	422	proportionalOnly	428		
externalLeading	422	resetButton	428		
fixedWidthOnly	427	returnValue	424		
fontFamily	422	setDefaultStyle	425		
fontWeight	422	setId	424		
fontWidth	423	vectorOnly	428		
helpButton	427	xHeight	423		
id	423	~IFontDialog	422		

The IFontDialog class displays 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 (Vol. IV) object on the IFontDialog::Settings constructor (p. 430). 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.

PM

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

IFontDialog

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.

Public Functions

Constructors

You can construct and destruct objects of this class, but you cannot copy them.

IFontDialog

❏	IFontDialog(IWindow* parent,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	IWindow* owner,	Y	Y	Y
	IHandler* handler,			
	const Style& style = defaultStyle (),			
	const Settings& settings = Settings (0));			

You can construct an object of this class by using this constructor, which accepts a parent, owner, handler, style, and settings as parameters.

To add a IFontDialogHandler (p. 435) to a modal IFontDialog, you must specify the handler on the constructor. Once you create a modal IFontDialog, 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 this constructor are the following:

- | | |
|----------------|---|
| <i>parent</i> | The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog. |
| <i>owner</i> | The owner window. |
| <i>handler</i> | A pointer to an IHandler (Vol. II) object. |
| <i>style</i> | Use the styles provided by IFontDialog (p. 426) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional. |

IFontDialog

settings Use the settings provided by IFontDialog::Settings (p. 430) to specify the setting. Optional.

```
2 IFontDialog( IWindow* parent,                                Win PM Motif
               IWindow* owner,                                Y  Y  Y
               const Style& style = defaultStyle ( ) );
```

You can construct an object of this class by using this constructor, which accepts a parent, owner, and style as parameters.

The parameters for this constructor 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 Use the styles provided by IFontDialog (p. 426) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional.

```
3 IFontDialog( IWindow* parent,                                Win PM Motif
               IWindow* owner,                                Y  Y  Y
               const Settings& settings,
               const Style& style = defaultStyle ( ) );
```

You can construct an object of this class by using this constructor, which accepts a parent, owner, settings, and style as parameters.

The parameters for this constructor 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.

settings Use the settings provided by IFontDialog::Settings (p. 430) to specify the setting.

style Use the styles provided by IFontDialog (p. 426) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style. Optional.

```
4 IFontDialog( IWindow* parent,                                Win PM Motif
               IWindow* owner,                                Y  Y  Y
               const Style& style,
               const Settings& settings );
```

IFontDialog

You can construct an object of this class by using this constructor, which accepts a parent, owner, style, and settings as parameters.

The parameters for this constructor are the following:

<i>parent</i>	The parent of the dialog box. If you specify 0, the desktop becomes the parent of the dialog.
<i>owner</i>	The owner window.
<i>style</i>	Use the styles provided by IFontDialog (p. 426) to specify the style. The default creates a modal dialog. To construct a modeless dialog, specify the modeless style.
<i>settings</i>	Use the settings provided by IFontDialog::Settings (p. 430) to specify the setting.

~IFontDialog

virtual		<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IFontDialog();		<i>Y</i>	<i>Y</i>	<i>Y</i>

Getting Information about the Chosen Font

Use these members to get information about the font that was selected in the font dialog.

emHeight Returns the height of the Em square for the font.

unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
emHeight() const;		<i>N</i>	<i>Y</i>	<i>Y</i>

externalLeading

Returns the amount of white space between lines of text.

unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
externalLeading() const;		<i>N</i>	<i>Y</i>	<i>Y</i>

fontFamily Returns the font's family name.

IString		<u>Win</u>	<u>PM</u>	<u>Motif</u>
fontFamily() const;		<i>N</i>	<i>Y</i>	<i>Y</i>

fontWeight Returns the weight class (boldness) of the font.

IFontDialog

unsigned long
fontWeight() const;

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



This function always returns 0.

fontWidth Returns the width class of the font.

unsigned long
fontWidth() const;

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



This function always returns 0.

nominalPointSize

For a bitmap font, the height of the font is returned. For a vector font, the optimal size for the font is returned.

unsigned long
nominalPointSize() const;

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



For scalable fonts, this function returns 0.

pointSize Returns the font's point size.

unsigned long
pointSize() const;

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

xHeight Returns the height above the baseline for lowercase letters.

unsigned long
xHeight() const;

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

Getting Information about the Dialog

Use these members to get information about the dialog.

buttonPressedId

Returns the ID of the push button used to end the dialog.

unsigned long
buttonPressedId() const;

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

id Returns the window identifier of the window.

IFontDialog

	unsigned long id() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
isModeless	If the font dialog is modeless, true is returned.			
	Boolean isModeless() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
pressedOK	If the user ended the dialog by selecting the OK push button, true is returned.			
	Boolean pressedOK() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
returnValue	If an error occurs, the return code is returned.			
	long returnValue() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

Setting Information about the Dialog

Use these members to set specific information about the dialog.

setId Sets the window identifier of the window.

IFontDialog& setId(unsigned long newIdentifier);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

Styles

The style members provide a set of valid font dialog styles for IFontDialog::setDefaultStyle and the constructors of the IFontDialog (p. 419) class.

convertToGUIStyle

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

virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 427) unless you have changed it using setDefaultStyle (p. 425).

IFontDialog

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

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

setDefaultStyle

Sets the default style for all subsequent font dialogs.

style Use the styles provided by IFontDialog (p. 426) to specify the default style.

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

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

Inherited Public Functions

IFrameWindow		
addExtension	isAnExtension	setBorderSize
addToWindowList	isFlashing	setBorderWidth
backgroundColor	isMaximized	setClient
beginFlashing	isMinimized	setDefaultOrdering
borderHeight	isModal	setDefaultStyle
borderSize	matchForMnemonic	setDestroyOnClose
borderWidth	maximize	setExtensionSize
client	maximizeRect	setIcon
clientHandle	minimize	setLayoutDistorted
clientRectFor	minimizeRect	setMousePointer
close	mousePointer	setRestoreRect
convertToGUIStyle	moveSizeToClient	setResult
defaultOrdering	nextShellRect	setToolBarList
defaultPushButton	notifyOwner	shareParentDBCSStatus
defaultStyle	removeExtension	show
disabledBackgroundColor	removeFromWindowList	showModally
dismiss	resetBackgroundColor	start
enableNotification	resetDisabledBackgroundColor	toolBarList
endFlashing	restore	topHandle
frameRectFor	restoreRect	update

IFontDialog

IFrameWindow		
handleFor	result	useExtensionMinimumSize
icon	setBorderHeight	usesDialogBackground

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

IFrameWindow		
addDefaultHandler	findExtensions	registerFrameClass
attachClient	initialize	removeDefaultHandler
create	isFrameWindow	setExtensions
extensions	isRelatedHandle	tryToLoadDialog
findExtension	registerCallbacks	unregisterCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Getting Information about the Dialog

Use these members to get information about the dialog.

cancel Returned push button ID value when the user dismisses the dialog by pressing the **Cancel** button.

IFontDialog

static const unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
cancel;	Y	Y	Y

ok Returned push button ID value when the user dismisses the dialog by pressing the **OK** button.

static const unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
ok;	Y	Y	Y

Styles

The style members provide a set of valid font dialog styles for IFontDialog::setDefaultStyle and the constructors of the IFontDialog (p. 419) class.

applyButton Adds an **Apply** push button to the dialog. This is useful in a modeless dialog.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
applyButton;	Y	Y	Y

bitmapOnly Specifies that the dialog present bitmap fonts only.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bitmapOnly;	Y	Y	Y

classDefaultStyle

Specifies the original default style for this class, which is a static variable set to 0 indicating no style flags are set.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

fixedWidthOnly

Specifies that the dialog present fixed-width (monospace) fonts only.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
fixedWidthOnly;	Y	Y	Y

helpButton Adds a **Help** push button to the dialog. This push button sends a help message to the owner of the dialog.

IFontDialog

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>helpButton;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

modeless Specifies that 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 (p. 437).

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>modeless;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

noStyle Disables all of the valid styles. Use this style only when you are not using any other styles.

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

noSynthesize Specifies the dialog does not manipulate bitmap fonts to synthesize features, such as italics or bold.

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>noSynthesize;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

proportionalOnly Specifies that the dialog present proportionally-spaced fonts only.

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>proportionalOnly;</code>	<i>I</i>	<i>Y</i>	<i>Y</i>

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.

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>resetButton;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

vectorOnly Specifies that the dialog present vector fonts only.

	<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<code>vectorOnly;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Data

IFrameWindow		
activateId	closeId	deactivateId

IWindow		
activeColorId	disabledForegroundColorId	hiliteForegroundColorId
backgroundColorId	enableId	inactiveColorId
borderColorId	focusId	positionId
commandId	fontId	shadowColorId
deleteId	foregroundColorId	sizeId
disabledBackgroundColorId	hiliteBackgroundColorId	systemCommandId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IFontDialog contains the following nested classes:

IFontDialog::Style (see page 434)

IFontDialog::Settings (see page 430)

IFontDialog::Settings

IFontDialog::Settings

Derivation

```
graph TD
    IBase --> IFontDialog_Settings[IFontDialog::Settings]
```

Inherited By None.

Header File ifontdlg.hpp

Members

Member	Page	Member	Page
Constructor	431	setPreviewText	431
setDialogTemplate	431	setPrinterPS	433
setDisplayPS	432	setSizeList	432
setFamily	432	Settings	431
setFont	432	setTitle	431
setPointSize	432	~Settings	431
setPosition	431		

The IFontDialog::Settings class contains input for an IFontDialog (p. 419) 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

PM

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

Public Functions

Constructors

You can construct and destruct object of this class. There is one constructor, the default, which accepts an optional IFont object. If you specify an IFont object, it is used as the initially selected font in the dialog. If the user selects the **OK** push button, the specified font is changed to the font selected by the user.

IFontDialog::Settings

Settings

You can construct an object of this class by using this constructor, which accepts an optional IFont object. If specified, this IFont object is used to specify the initial font to display in the dialog.

```
Settings( IFont* font = 0 );
```

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

~Settings

```
~Settings();
```

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

Dialog Customization

Use these members to customize the font dialog presented to the user. This includes the dialog's title, font preview text, dialog position, and the dialog template to use.

setDialogTemplate

Sets a dialog template resource to use in place of the OS/2-supplied default font dialog.

```
Settings&  
    setDialogTemplate( const IResourceId& templateId );
```

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

setPosition

Sets the initial placement of the dialog.

```
Settings&  
    setPosition( const IPoint& position );
```

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

setPreviewText

Sets the text to display in the font sample box.

```
Settings&  
    setPreviewText( const char* previewText );
```

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

setTitle

Sets the font dialog's title.

```
1 Settings&  
    setTitle( const IResourceId& text );
```

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

IFontDialog::Settings

2	Settings& setTitle(const char* title);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
----------	---	-----------------	----------------	-------------------

Setting Information about the Initial Font

Use these members to set information about the initial font that is displayed in the font dialog.

setFamily Sets the font family name for the initial set of fonts displayed in the font dialog.

Settings& setFamily(const char* fontFamily);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

setFont Sets the font to be displayed as the initial selection in the dialog.

Settings& setFont(IFont* font);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--------------------------------------	-----------------	----------------	-------------------

setPointSize Sets the point size for the initial set of fonts displayed in the font dialog.

Settings& setPointSize(unsigned long pointSize);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------



If you specify a point size of 0, X-scalable fonts are displayed.

setSizeList Sets the list of point size choices.

Settings& setSizeList(const char* sizeList);	<u>Win</u> N	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------



This only affects dialogs for vector fonts in an OS/2 operating system. A dialog for an OS/2 operating system bitmap font has the size list automatically set to the available sizes for the bitmap font.



This function has no effect in Motif.

Setting the Presentation Space

Use these members to set the presentation space that is used to determine which fonts are available.

setDisplayPS Sets the display's presentation space. The dialog uses this presentation space to determine which fonts are available.

Settings& setDisplayPS(const IPresSpaceHandle& presSpaceHandle);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> I
---	-----------------	----------------	-------------------

IFontDialog::Settings

setPrinterPS Sets the printer's presentation space. The dialog uses this presentation space to determine which fonts are available for the printer.

```
Settings&  
    setPrinterPS( const IPresSpaceHandle& presSpaceHandle );
```

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

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFontDialog::Style

IFontDialog::Style

Derivation IBase
IBitFlag
IFontDialog::Style

Inherited By None.

Header File ifontdlg.hpp

The nested class IFontDialog::Style provides a set of valid styles for the IFontDialog (p. 419) class.



In Motif, the style vectorOnly specifies that the dialog present scalable fonts only. AIX does not support the noSynthesize style.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IFontDialogHandler

Derivation IBase
IVBase
IHandler
IFontDialogHandler

Inherited By None.

Header File ifonthdr.hpp

Members	Member	Page	Member	Page
	Constructor	436	modelessResults	437
	dispatchHandlerEvent	437	~IFontDialogHandler	436
	modelessFontApplied	437		

The IFontDialogHandler class handles 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 (Vol. II) 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 (p. 419) constructor.

When the font dialog handler receives a font dialog event, it creates an IFontDialog object and routes that object to the IFontDialogHandler::modelessResults (p. 437) 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.

IFontDialogHandler

- If this is the last handler for the font dialog, call IWindow::defaultProcedure (Vol. II) to process the font dialog event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IFontDialogHandler

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

IFontDialogHandler();

Win
Y

PM
Y

Motif
Y

~IFontDialogHandler

virtual
~IFontDialogHandler();

Win
Y

PM
Y

Motif
Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Dispatch Events

These members determine if the event is one of the font dialog events. If it is, it calls the appropriate function.

dispatchHandlerEvent

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

virtual Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
dispatchHandlerEvent(IEvent& event);		Y	Y	Y

Font Dialog Events

These members are called when a file dialog event occurs. Override these to provide your own processing.

modelessFontApplied

Called when the apply button of a modeless dialog is pressed. Allows you to update an application based on the current setting of the dialog without closing the dialog.

appliedDialog

Use the IFontDialog* pointer to get additional information about the dialog.

appliedFont

The font that is currently selected in the dialog. This is the IFont object that was passed into the dialog in a settings object and has been modified based on the current settings of the dialog.

virtual Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
modelessFontApplied(IFontDialog* appliedDialog, IFont* appliedFont);		Y	Y	I

modelessResults

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 (p. 419) if needed. Use IFontDialog::pressedOK (p. 424) 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:

IFontDialogHandler

```
endingDialog->setAutoDeleteObject().
```

```
virtual Boolean
modelessResults( IFontDialog* endingDialog );
```

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

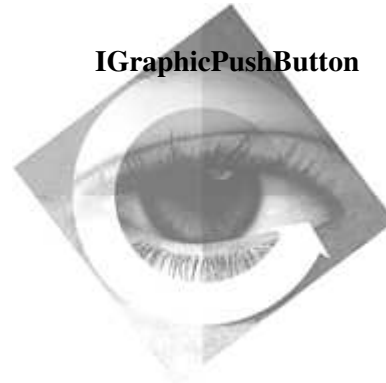
Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IGraphicPushButton



IGraphicPushButton

Derivation

```

IBase
  IVBase
    INotifier
      IWindow
        IControl
          ITextControl
            IButton
              IPushButton
                IGraphicPushButton
  
```

Inherited By None.

Header File igrphbt.hpp

Members	Member	Page	Member	Page
	Constructor	440	icon	443
	bitmap	442	isSizeToGraphic	444
	calcMinimumSize	446	marginSize	444
	classDefaultStyle	447	moveSizeTo	445
	convertToGUIStyle	444	setDefaultStyle	445
	currentGraphicType	442	setGraphic	443
	defaultStyle	445	setMarginSize	444
	disableSizeToGraphic	443	sizeToGraphic	447
	enableSizeToGraphic	444	~IGraphicPushButton	442
	graphicWindow	442		

The IGraphicPushButton class creates and manages graphic push button control windows. The standard push button generates an ICommandEvent (Vol. II). 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 (Vol. II) and IPushButton::enableSystemCommand (Vol. II). These functions add or remove the styles IPushButton::help (Vol. II) and IPushButton::systemCommand (Vol. II), respectively. For example:

```

IGraphicPushButton pbCancel(ID_CANCEL, this, this, ID_SOMEBMP,
                             IRectangle(10,10,80,22));
  
```



The IGraphicPushButton control consists of an XmPushButton widget. The XmNlabelPixmap resource is set to the graphic specified on the constructor.



AIX does not support dialog templates.

IGraphicPushButton

Public Functions

Constructors

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

IGraphicPushButton

1

IGraphicPushButton(
 unsigned long id,
 IWindow* parent,
 IWindow* owner,
 const IResourceId& bitmapOrIconId,
 const IRectangle& initial = IRectangle (),
 const Style& style = defaultStyle ());

Win

PM

Motif

Y

Y

Y

Creates a graphic push button using the specified window ID, parent and owner windows, screen position and size, resource library with its specified icon or bitmap ID, and window style. If you have not already loaded the bitmap or icon and you want to load it from the resource library of your choice, use this constructor.

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

2

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

Win

PM

Motif

Y

Y

Y

Creates 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 bitmap ID, and window style. If you have not already loaded the bitmap or icon and you want to load it from the default resource library, use this constructor.

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

IGraphicPushButton

3 IGraphicPushButton(Win PM Motif
 unsigned long id, Y Y Y
 IWindow* parent,
 IWindow* owner,
 const IBitmapHandle& bitmap = IBitmapHandle (),
 const IRectangle& initial = IRectangle (),
 const Style& style = defaultStyle ());

Creates a graphic push button with the specified window ID, parent and owner windows, screen position and size, bitmap, and window style. If you already have a bitmap handle, use this constructor to put the bitmap on the graphic push button.

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

4 IGraphicPushButton(Win PM Motif
 unsigned long id, Y Y Y
 IWindow* parent,
 IWindow* owner,
 const IPointerHandle& icon,
 const IRectangle& initial = IRectangle (),
 const Style& style = defaultStyle ());

Creates 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.

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

5 IGraphicPushButton(unsigned long id, Win PM Motif
 IWindow* parent); Y Y Y

Creates a graphic push button object for the specified push button in a dialog template that has the icon or bitmap style.

Exceptions	
InvalidParameter	The parent window pointer specified was invalid. You must specify a valid IWindow pointer as the parent.
IAccessError	Control information could not be obtained from the system for the specified window. Verify that the specified ID and parentDialog combination refer to a valid push button control.

IGraphicPushButton

6 IGraphicPushButton(const IWindowHandle& handle); Win PM Motif
Y Y Y

Creates a graphic push button object for an existing bitmap or icon push button control with the specified window handle.

Exceptions	
IAccessError	Control information could not be obtained from the system for the specified window. Verify that the specified handle refers to a valid push button control.

~IGraphicPushButton

virtual Win PM Motif
Y Y Y
~IGraphicPushButton();

Graphic

A *graphic* is an icon or bitmap that is displayed on a graphic push button. You can query the type, handle, or window of the current graphic and set a new graphic.

bitmap Returns the handle of the currently set bitmap. If no bitmap is set into the graphic push button, an IBitmapHandle (Vol. II) is returned.

IBitmapHandle Win PM Motif
Y Y Y
bitmap() const;

currentGraphicType

Returns the current type of graphic set onto the graphic push button. The returned value is an enumerator provided by IGraphicPushButton::GraphicType (p. 448).

GraphicType Win PM Motif
Y Y Y
currentGraphicType() const;

graphicWindow

Returns the IIconControl (p. 451) used for drawing the graphic.

IIconControl& Win PM Motif
Y Y Y
graphicWindow() const;



AIX does not support this function; therefore, it has no effect.

IGraphicPushButton

icon Returns the handle of the currently set icon. If no icon is set into the graphic push button, an IIconControl (p. 451) is returned.

IPointerHandle icon() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

setGraphic Changes the stored graphic to the specified value.

1 virtual IGraphicPushButton& setGraphic(const IPointerHandle& handle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Use this overload if you already have an icon handle for the icon you want to place on the graphic push button.

2 virtual IGraphicPushButton& setGraphic(unsigned long bitmapOrIconId);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Use this overload if you have not already loaded the bitmap or icon and you want to load it from the default resource library.

3 virtual IGraphicPushButton& setGraphic(const IResourceId& bitmapOrIconId);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Use this overload if you have not already loaded the bitmap or icon and you want to load it from the resource library of your choice.

4 virtual IGraphicPushButton& setGraphic(const IBitmapHandle& handle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Use this overload if you already have a bitmap handle for the bitmap you want to place on the graphic push button.

Graphic Size

Use these style members to query and modify the style of a graphic push button object.

disableSizeToGraphic

Removes the style sizeToGraphic (p. 447).

virtual IGraphicPushButton& disableSizeToGraphic();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	I



AIX does not support this function; therefore, it has no effect.

IGraphicPushButton

enableSizeToGraphic

Adds or removes the style `sizeToGraphic` (p. 447).

<pre>virtual IGraphicPushButton& enableSizeToGraphic(Boolean sizeToGraphic = true);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					



AIX does not support this function; therefore, it has no effect.

isSizeToGraphic

If the style `sizeToGraphic` (p. 447) is set, `true` is returned.

<pre>Boolean isSizeToGraphic() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Margins

The *margin* is the area between the graphic and the outside border of the push button. If you have not set the `sizeToGraphic` style, you can change the size of this area.

marginSize Returns the currently set margin size.

<pre>ISize marginSize() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

setMarginSize

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.

<pre>IGraphicPushButton& setMarginSize(const ISize& size);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>I</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>I</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>I</i>					

Styles

These style members provide valid styles for `IGraphicPushButton::setDefaultStyle` and for the constructor of the `IGraphicPushButton` (p. 439) class.

convertToGUIStyle

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

IGraphicPushButton

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

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

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 447) unless you have changed it using setDefaultStyle (p. 445).

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

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

setDefaultStyle

Sets the default style for all subsequent graphic push buttons. Use the styles provided by IGraphicPushButton (p. 447) to specify the default style.

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

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

Window Positioning

Use these members to manipulate the size and position of windows.

moveSizeTo Changes the position and size of the window.

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

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

Inherited Public Functions

IPushButton		
addBorder	disableSystemCommand	isDefault
convertToGUIStyle	enableDefault	isHelp
defaultStyle	enableHelp	isSystemCommand
disableDefault	enableSystemCommand	removeBorder
disableHelp	hasBorder	setDefaultStyle

IButton		
allowsMouseClickedFocus	disableMouseClickedFocus	highlight
backgroundColor	enableMouseClickedFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor

IGraphicPushButton

IButton		
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout is information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the recommended minimum size of this graphic push button control. The size is based on the actual size of the stored graphic.

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

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

Inherited Protected Functions

IPushButton		
calcMinimumSize	passEventToOwner	registerCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

These style members provide valid styles for IGraphicPushButton::setDefaultStyle and for the constructor of the IGraphicPushButton (p. 439) class.

classDefaultStyle

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

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

sizeToGraphic

Displays the graphic on 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 on the push button. If you do not specify this style, the graphic is stretched to fill the push button within its border.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sizeToGraphic;	Y	Y	Y

Inherited Public Data

IPushButton		
classDefaultStyle	defaultButton	help

IGraphicPushButton

IButton		
buttonClickId	noPointerFocus	

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IGraphicPushButton contains the following nested classes:

IGraphicPushButton::Style (see page 450)

```
GraphicType  GraphicType {  
    bitmapType,  
    iconType  
};
```

IGraphicPushButton

These enumerators specify the current type of graphic set onto the graphic push button:

bitmapType

Specifies that a bitmap is set onto the graphic push button. If you do not specify a handle or resource ID when constructing the graphic push button object, an IBitmapHandle (Vol. II) is the default graphic. This is the default style.

iconType

Specifies an icon is set onto the graphic push button.

IGraphicPushButton::Style

IGraphicPushButton::Style

Derivation

IBase
IBitFlag
IGraphicPushButton::Style

Inherited By None.

Header File `igraphbt.hpp`

The nested class IGraphicPushButton::Style provides a set of valid styles for the IGraphicPushButton (p. 439) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IIconControl

Derivation	IBase
	IVBase
	INotifier
	IWindow
	IControl
	ITextControl
	IStaticText
	IBitmapControl
	IIconControl

Inherited By None.

Header File iiconctl.hpp

Members	Member	Page	Member	Page
	Constructor	452	moveSizeTo	455
	calcMinimumSize	456	setDefaultStyle	455
	classDefaultStyle	457	setIcon	454
	convertToGUIStyle	454	sizeToIcon	457
	defaultStyle	454	visibleRectangle	451
	icon	453	~IIconControl	453

The IIconControl class creates and operates upon icon controls. You can also use this class to add text to an icon by calling ITextControl::setText (Vol. II).



AIX does not support system bitmaps and dialog templates.

Public Functions

Canvas Support

These members override the inherited visibleRectangle function.

visibleRectangle

Returns a rectangle that represents the area of the window that is actually painted by the icon control.

virtual IRectangle	Win	PM	Motif
visibleRectangle() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

IIconControl

Constructors

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

IIconControl

1

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

Win

PM

Motif

Y

Y

Y

Create an icon control using the specified window ID, parent and owner windows, resource library with its specified icon ID, screen position and size, and window style. If you have not already loaded the resource, use this constructor.

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

2

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

Win

PM

Motif

Y

Y

Y

Create an icon control with the specified ID, parent and owner windows, default resource library with its specified icon ID, and window style. If you have not already loaded the icon and you want to load it from the default resource library, use this constructor.

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

3

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

Win

PM

Motif

Y

Y

Y

IIconControl

Create an icon control with the specified window ID, parent and owner windows, icon, screen position and size, and window style. If you already have an icon handle, use this constructor.

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

4 IIconControl(unsigned long id, Win PM Motif
IWindow* parent, Y Y N
IWindow* owner,
ISystemPointerHandle::Identifier icon,
const IRectangle& initial = IRectangle (),
const Style& style = defaultStyle ());

Create an icon control with the specified window ID, parent and owner windows, specified system icon, screen position and size, and window style. If you want to use a system icon, use this constructor.

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

5 IIconControl(unsigned long id, Win PM Motif
IWindow* parentDialog); Y Y Y

Create an IIconControl object for the specified icon from the dialog control.

6 IIconControl(const IWindowHandle& handle); Win PM Motif
Y Y Y

Create an IIconControl object for the specified icon window handle.

~IIconControl

virtual Win PM Motif
~IIconControl(); Y Y Y

Icons

Use these functions to query and set the icon image in an icon control window.

icon Returns the handle of the currently set icon.

IIconControl

	<code>IPointerHandle icon() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
setIcon	Sets the icon image of the icon control window.			
1	<code>virtual IIconControl& setIcon(const IPointerHandle& handle);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
	Set the icon using an existing icon handle.			
2	<code>virtual IIconControl& setIcon(unsigned long iconId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
	Set the icon using the default resource library with the specified icon ID.			
3	<code>virtual IIconControl& setIcon(const IResourceId& iconId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
	Set the icon by specifying an IResourceId. If you want to load the icon from a specific resource library, use this function.			
4	<code>virtual IIconControl& setIcon(ISystemPointerHandle::Identifier icon);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
	Set the icon using a system icon.			

Styles

These style members provide valid styles for IIconControl::setDefaultStyle and for the constructor of the IIconControl (p. 451) class.

convertToGUIStyle

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

	<code>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
defaultStyle	Returns the default style. This is classDefaultStyle (p. 457) unless you have changed it using setDefaultStyle (p. 455).			

IIconControl

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

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

setDefaultStyle

Sets the default style for all subsequent icon controls.

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

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

Window Positioning

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

moveSizeTo Changes the position and size of the window.

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

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



This IWindow function is overridden to handle icon sizing and positioning in Motif.

Inherited Public Functions

IBitmapControl		
bitmap	defaultStyle	setBitmap
convertToGUIStyle	moveSizeTo	setDefaultStyle

IStaticText		
alignment	enableFillBackground	isStrikeout
backgroundColor	enableHalftone	isUnderscore
convertToGUIStyle	enableStrikeout	limit
defaultStyle	enableUnderscore	moveSizeTo
disableFillBackground	fillColor	resetFillColor
disableHalftone	foregroundColor	setAlignment
disableStrikeout	hasFillBackground	setDefaultStyle
disableUnderscore	isHalftone	setFillColor

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IconControl

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout is information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the minimum size this icon control can be, based on the actual size of the currently set icon.

virtual ISize
calcMinimumSize() const;

Win

PM

Motif

Y

Y

Y

Exceptions	
IAccessError	The call to determine the size of the icon failed. The icon may be corrupted.

Inherited Protected Functions

IBitmapControl		
calcMinimumSize		

IIconControl

IStaticText		
calcLimitSize	calcMinimumSize	passEventToOwner

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

These style members provide valid styles for IIconControl::setDefaultStyle and for the constructor of the IIconControl (p. 451) class.

classDefaultStyle

Original default style for this class, which is IWindow::visible.

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

sizeToIcon Sizes the window to the size of the icon passed in on the constructor.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
sizeToIcon;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Data

IStaticText		
border3D	fillColorId	right
bottom	halftone	strikeout
center	halftoneId	strikeoutId
classDefaultStyle	left	top
fillBackground	limitId	underscore
fillBackgroundId	mnemonic	underscoreId

ITextControl		
textId		

IIconControl

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IIconControl contains the following nested classes:

IIconControl::Style (see page 459)



IIconControl::Style

Derivation IBase
 IBitFlag
 IIconControl::Style

Inherited By None.

Header File iiconctl.hpp

The nested class IIconControl::Style provides a set of valid styles for the IIconControl (p. 451) class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File iinfoa.hpp

Members				
Member	Page	Member	Page	
Constructor	461	missingTextId	468	
calcMinimumSize	467	resourceLibrary	463	
disabledText	463	resourceLibraryId	468	
disabledTextId	468	setDisabledText	464	
dispatchHandlerEvent	466	setInactiveText	464	
handleEventsFor	462	setLineCount	462	
inactiveText	463	setMissingText	464	
inactiveTextId	468	setResourceLibrary	463	
informationFor	466	setStringTableOffset	465	
lineCount	462	stopHandlingEventsFor	462	
menuEnded	467	stringTableOffset	465	
menuSelected	467	~IInfoArea	461	
missingText	464			

The IInfoArea class provides 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.



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 default user application resource library supplies the resources.



The User Interface Class Library loads the strings from the user default library.

IInfoArea

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.

Public Functions

Constructors

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

IInfoArea

1	<code>IInfoArea(IFrameWindow* frame, const char* resDLLName, unsigned long id = 0);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Use this constructor to create IInfoArea objects from a pointer to a frame window, the name of the resource DLL, and the ID for the information area control,

2	<code>IInfoArea(IFrameWindow* frame, unsigned long id = 0);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Use this constructor to construct an IInfoArea object from a frame window pointer and an ID for the information area control. By default, the information strings will be loaded from the user default resource library obtained from userResourceLibrary.

3	<code>IInfoArea(IFrameWindow* frame, unsigned long id, const char* resDLLName);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Use this constructor to create IInfoArea objects from a pointer to a frame window, an ID for the information area control, and the name of the resource DLL.

4	<code>IInfoArea(IFrameWindow* frame, const IModuleHandle& resMod, unsigned long id = 0);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

Use this constructor to create IInfoArea objects from a pointer to a frame window, a module handle of the resource DLL, and an ID for the information area control. If the module handle is 0, the information strings are loaded from the current .EXE.

~IInfoArea

IInfoArea

```
virtual  
    ~IInfoArea();
```

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

Display Size

Use these members to control the height of the information area's minimum size, in terms of lines of text. By default, the frame uses this to size the height of this frame extension.

lineCount Returns the number of text lines that the information area uses.

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

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

setLineCount Sets the number of text lines that the information area uses to display text. The default is one line.

```
virtual IInfoArea&  
    setLineCount( unsigned long lines = 1 );
```

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

Exceptions	
InvalidParameter	The number of lines specified must be greater than 0.

Event Dispatching

Use these members to attach the handler of the information area to and detach it from a frame window. These members are used when adding the information area extension to the frame window, so you should not have to call them.

handleEventsFor

Attaches the handler of the information area to the specified frame window.

```
virtual IInfoArea&  
    handleEventsFor( IFrameWindow* frame );
```

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

Exceptions	
InvalidParameter	An invalid frame window pointer was specified. You must specify a valid IFrameWindow pointer.

stopHandlingEventsFor

Detaches the handler of the information area from the specified frame window.

```
virtual IInfoArea&  
    stopHandlingEventsFor( IFrameWindow* frame );
```

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

IInfoArea

Exceptions	
InvalidParameter	An invalid frame window pointer was specified. You must specify a valid IFrameWindow pointer.

Resource Library

Use these members to query and set the resource library from which the information strings are to be loaded.

resourceLibrary

Returns a reference to the library being used.

<code>virtual IResourceLibrary& resourceLibrary() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

setResourceLibrary

Sets the specified resource library.

1	<code>virtual IInfoArea& setResourceLibrary(const char* resDLLName);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

Use this function to set the resource library using the name of a .DLL file.

2	<code>virtual IInfoArea& setResourceLibrary(const IModuleHandle& resMod);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

Use this function to set the resource library using a module handle.

Special Information Strings

Use these members to set and query the value of the information strings displayed in special circumstances. Special circumstances include when the menu is inactive, when a disabled menu item is the current menu item, or when the text for the selected menu item is missing. Each special string can be specified as either an IString or the identifier for a string from the resource library.

disabledText Returns the text displayed when the menu item at the selection cursor is disabled.

<code>virtual IString disabledText() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

inactiveText Returns the text displayed when the menu is inactive.

IInfoArea

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

missingText Returns the text that displays when the User Interface Class Library cannot obtain the required information string from the resource library.

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

setEnabledText

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.

1 virtual IInfoArea& setEnabledText(const IString& disabledText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

2 virtual IInfoArea& setEnabledText(unsigned long disabledText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

setInactiveText

Sets the text that is used when the menu is inactive.

1 virtual IInfoArea& setInactiveText(unsigned long inactiveText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

2 virtual IInfoArea& setInactiveText(const IString& inactiveText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

setMissingText

Sets the text that displays when the User Interface Class Library cannot find the information string.

1 virtual IInfoArea& setMissingText(unsigned long missingText);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

IInfoArea

2	<pre>virtual IInfoArea& setMissingText(const IString& missingText);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

String Table Offset

Use these members to set and query the string table offset. The string table offset is added to the menu identifier to obtain the identifier for the string providing the information about the menu item. The default is zero.

setStringTableOffset

Sets the offset to be used to calculate the information string ID.

<pre>virtual IInfoArea& setStringTableOffset(long newOffset = 0);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

stringTableOffset

Returns the offset currently in use.

<pre>virtual long stringTableOffset() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Inherited Public Functions

IStaticText		
alignment	enableFillBackground	isStrikeout
backgroundColor	enableHalftone	isUnderscore
convertToGUIStyle	enableStrikeout	limit
defaultStyle	enableUnderscore	moveSizeTo
disableFillBackground	fillColor	resetFillColor
disableHalftone	foregroundColor	setAlignment
disableStrikeout	hasFillBackground	setDefaultStyle
disableUnderscore	isHalftone	setFillColor

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

InfoArea

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Event Dispatching

Use these members to attach the handler of the information area to and detach it from a frame window. These members are used when adding the information area extension to the frame window, so you should not have to call them.

dispatchHandlerEvent

Calls the appropriate virtual function or functions to process an information area event.

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

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

Information String Processing

These members implement this class. Derived classes can override them to tailor the behavior.

informationFor

Returns the information string for the specified menu ID.

IInfoArea

<pre>virtual IString informationFor(unsigned long menuId) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Layout Support

Layout is information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

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.

<pre>virtual ISize calcMinimumSize() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

Menu Event Processing

These members are overridden from class IMenuHandler to handle the menu events appropriately.

menuEnded Places the inactive information string into the information area.

<pre>virtual Boolean menuEnded(IMenuEvent& menuEvent);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

menuSelected

Places the corresponding information string into the information area.

<pre>virtual Boolean menuSelected(IMenuEvent& menuEvent);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Inherited Protected Functions

IStaticText		
calcLimitSize	calcMinimumSize	passEventToOwner

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

IInfoArea

Public Data

Notification Members

These INotificationId strings are used for all notifications that IInfoArea provides to its observers.

disabledTextId

Notification identifier provided to observers when the disabled text of the information area control changes. IInfoArea provides a pointer to the new text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
disabledTextId;	<i>Y</i>	<i>Y</i>	<i>Y</i>

inactiveTextId

Notification identifier provided to observers when the inactive text of the information area control changes. IInfoArea provides a pointer to the new text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
inactiveTextId;	<i>Y</i>	<i>Y</i>	<i>Y</i>

missingTextId

Notification identifier provided to observers when the missing text of the information area control changes. IInfoArea provides a pointer to the new text string in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
missingTextId;	<i>Y</i>	<i>Y</i>	<i>Y</i>

resourceLibraryId

Notification identifier provided to observers when the resource library of the information area control changes. IInfoArea provides a pointer to the new resource library in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I).

static INotificationId const	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resourceLibraryId;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Data

IStaticText		
border3D	fillColorId	right
bottom	halftone	strikeout
center	halftoneId	strikeoutId
classDefaultStyle	left	top
fillBackground	limitId	underscore
fillBackgroundId	mnemonic	underscoreId

ITextControl		
textId		

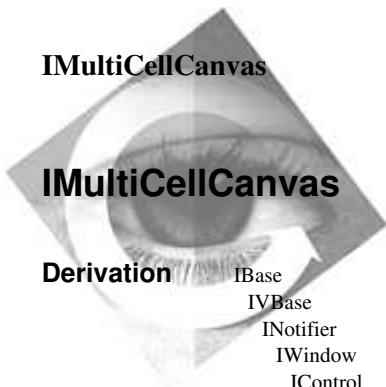
IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IMultiCellCanvas

IMultiCellCanvas

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ICanvas
IMultiCellCanvas

Inherited By IMMPlayerPanel

Header File imcelcv.hpp

Members

Member	Page	Member	Page
Constructor	476	hasGridLines	478
addToCell	474	isColumnExpandable	479
classDefaultStyle	484	isRowExpandable	480
columnWidth	479	layout	484
convertToGUIStyle	482	removeFromCell	475
defaultCell	479	rowHeight	480
defaultStyle	482	setColumnWidth	480
disableDragLines	478	setDefaultCell	481
disableGridLines	478	setDefaultStyle	482
dragLines	485	setLayoutDistorted	479
enableDragLines	478	setRowHeight	481
enableGridLines	478	windowInCell	476
gridLines	485	~IMultiCellCanvas	477
hasDragLines	478		

The IMultiCellCanvas class manages a collection of child windows in a way similar to that of data stored in a spreadsheet. A multicell canvas is a two-dimensional array of *cells* with an origin of (1,1) in the upper-left corner. Windows are added to a multicell 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:

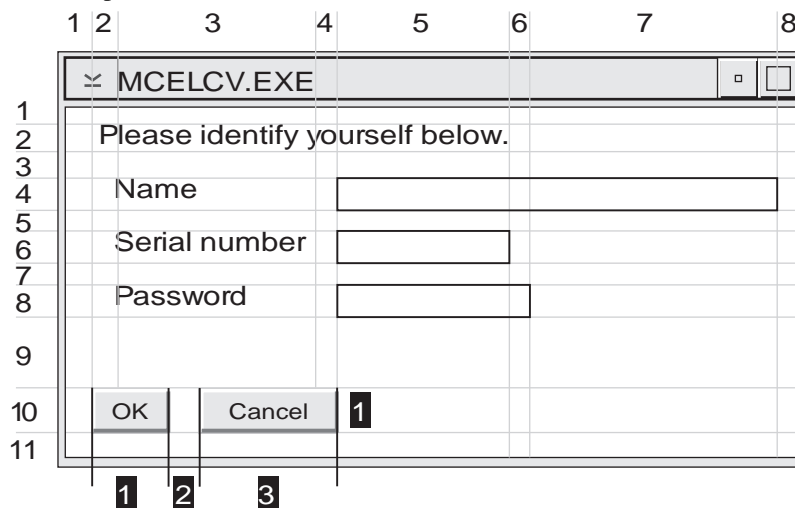
- Calling setMinimumSize
- Creating a derived class and implementing a calcMinimumSize function

IMultiCellCanvas

The user cannot resize fixed rows or columns by changing the size of the multicell canvas.

Undesirable movement of the windows might result when a multicell 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:



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 multicell 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 multicell 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

IMultiCellCanvas

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

IMultiCellCanvas

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 nonexpandable columns. The final value, true, indicates that this column is expandable.

Positioning Push Buttons Example

The last part of this multicell canvas is actually another multicell canvas. The two push buttons, **OK** and **Cancel**, are contained in a multicell canvas that is nested inside the multicell canvas that contains the other controls. To do this, we first defined a nested multicell canvas called `myButtons` and added it to the bottom of the `myClient` multicell 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 multicell 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());
```

IMultiCellCanvas

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.



You can create child windows of an IMultiCellCanvas which are not added to any cells of the IMultiCellCanvas. The IMultiCellCanvas, however, will only perform positioning of child controls that are added to the canvas. Therefore, the positioning of child controls that are not added to the canvas may vary from platform to platform because coordinate system differences are not handled automatically.

You can avoid this problem by ensuring that all visible child controls are added to the canvas.

Public Functions

Cell Contents

Use these members to position child windows of the canvas relative to one another, in cells.

addToCell 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 multicell canvas has no windows. Therefore, you must call this function to add windows to the canvas.

Note: If the multicell canvas is already shown, this function does not cause it to update the canvas itself immediately. You must call `IWindow::refresh` (Vol. II) to update the appearance of the canvas.

You do not specify how many rows and columns a multicell canvas contains. IMultiCellCanvas determines the maximum number of rows and columns from `addToCell`, `setColumnWidth` (p. 480), and `setRowHeight` (p. 481).

IMultiCellCanvas

```
virtual IMultiCellCanvas&
  addToCell( IWindow* childWindow,
             unsigned long startingColumn,
             unsigned long startingRow,
             unsigned long numberOfColumns = 1,
             unsigned long numberOfRows = 1 );
```

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

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 IMultiCellCanvas throws an IInvalidParameter exception when it runs its layout routine.

startingColumn

The leftmost column where the child window is placed. Columns in a multicell canvas are numbered beginning with 1.

startingRow

The topmost row where the child window is placed. Rows in a multicell canvas are numbered beginning with 1.

numberOfColumns

Number of columns that the child window will span. The default value is 1. Optional.

numberOfRows

Number of rows that the child window will span. The default value is 1. Optional.

Exceptions	
IInvalidRequest	The canvas already contains a child window that begins in the cell identified by <i>startingColumn</i> and <i>startingRow</i> .

removeFromCell

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 multicell canvas is already shown, this function does not cause it to update immediately. You must call IWindow::refresh (Vol. II) to update the appearance of the canvas.

IMultiCellCanvas

1	<pre>virtual IWindow* removeFromCell(unsigned long column, unsigned long row);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

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 (p. 474) 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 (p. 474) call that added the child window to the canvas.

2	<pre>virtual IWindow* removeFromCell(IWindow* childWindow);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

childWindow
 The child window to remove from the multicell canvas.

windowInCell Returns the child window occupying the specified cell.

<pre>virtual IWindow* windowInCell(unsigned long startingColumn, unsigned long startingRow) const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

startingColumn
 Identifier for the column. Columns in a multicell canvas are numbered beginning with 1.

startingRow
 Identifier for the row. Rows in a multicell canvas are numbered beginning with 1.

Constructors

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

IMultiCellCanvas

<pre>IMultiCellCanvas(unsigned long windowIdentifier, IWindow* parent, IWindow* owner, const IRectangle& initialSize = IRectangle (), const Style& style = defaultStyle ());</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

IMultiCellCanvas

windowIdentifier

The window identifier of the canvas you are constructing.

We recommend that you do the following:

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

Presentation Manager Notes: Do not use FID_xxx values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

parent

The parent window of the canvas you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.

owner

The owner window of the canvas you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.

Motif Notes: The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initialSize

The initial position and size of the canvas you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.

style

The window's characteristics. This value can be a combination of IMultiCellCanvas::Style (p. 484), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects. Optional.

~IMultiCellCanvas

virtual

~IMultiCellCanvas();

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

Grid and Drag Lines

These members provide visual debugging aids for you to use in your IMultiCellCanvas objects.

IMultiCellCanvas

disableDragLines

Removes the style IMultiCellCanvas::dragLines (p. 485) from a multicell canvas.

virtual IMultiCellCanvas& disableDragLines();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

disableGridLines

Removes the style IMultiCellCanvas::gridLines (p. 485) from a multicell canvas.

virtual IMultiCellCanvas& disableGridLines();	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

enableDragLines

Sets the style IMultiCellCanvas::dragLines (p. 485) for a multicell canvas.

virtual IMultiCellCanvas& enableDragLines(Boolean enable = true);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

enableGridLines

Sets the style IMultiCellCanvas::gridLines (p. 485) for a multicell canvas.

virtual IMultiCellCanvas& enableGridLines(Boolean enable = true);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
--	------------------------	-----------------------	--------------------------

hasDragLines

Queries whether a multicell canvas has the style IMultiCellCanvas::dragLines (p. 485) set.

Boolean hasDragLines() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
----------------------------------	------------------------	-----------------------	--------------------------

hasGridLines Queries whether a multicell canvas has the style IMultiCellCanvas::gridLines (p. 485) set.

Boolean hasGridLines() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>I</i>
----------------------------------	------------------------	-----------------------	--------------------------

Layout Support

Use layout members to determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

IMultiCellCanvas

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.

The internal state of a window is indicated by the Layout (Vol. II) 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

virtual IMultiCellCanvas& setLayoutDistorted(unsigned long layoutAttributesOn, unsigned long layoutAttributesOff);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

Row and Column Sizing

Use these members to query and set the width of columns and the height of rows in the canvas, as well as to set their ability to expand with the canvas.

columnWidth Returns the current width of the specified column in pixels.

unsigned long columnWidth(unsigned long column) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

column Identifier of the column. Columns in a multicell canvas are numbered beginning with 1.

defaultCell Queries the current default cell size. The initial default size is 10 pixels by 10 pixels. You can change it by calling the function setDefaultCell (p. 481).

static ISize defaultCell();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--------------------------------	-----------------	----------------	-------------------

isColumnExpandable

If the specified column can be expanded when the canvas is larger than its minimum size, true is returned. You can use setColumnWidth (p. 480) to make a column expandable.

IMultiCellCanvas

Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>isColumnExpandable(unsigned long column) const;</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

column Identifier for the column. Columns in a multicell canvas are numbered beginning with 1.

isRowExpandable

If the specified row can be expanded when the canvas is larger than its minimum size, true is returned. You can use `setRowHeight` (p. 481) to make a row expandable.

Boolean		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>isRowExpandable(unsigned long row) const;</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

row Identifier for the row. Rows in a multicell canvas are numbered beginning with 1.

rowHeight Returns the current height of the specified row in pixels.

unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>rowHeight(unsigned long row) const;</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

row Identifier for the row. Rows in a multicell canvas are numbered beginning with 1.

setColumnWidth

Defines the smallest width that the column will have and indicates whether the column is expandable. 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 multicell canvas contains. `IMultiCellCanvas` determines the maximum number of rows and columns from `addToCell` (p. 474), `setColumnWidth`, and `setRowHeight` (p. 481).

Note: If the multicell canvas is already shown, this function does not cause it to update immediately. You must call `IWindow::refresh` (Vol. II) to update the appearance of the canvas.

IMultiCellCanvas

```
virtual IMultiCellCanvas&
    setColumnWidth( unsigned long column,
                    unsigned long widthInPixels,
                    Boolean expandable = false );
```

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

column Identifier for the column. Columns in a multicell canvas are numbered beginning with 1.

widthInPixels Minimum width in pixels.

expandable If set to true, windows in the column are expanded horizontally when the canvas is larger than its minimum size. The default is false. Optional.

setDefaultCell

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 multicell canvas to update the canvas itself immediately. To force a multicell canvas to update after calling this function, call `setLayoutDistorted` (p. 479), as follows:

```
setLayoutDistorted(IWindow::layoutChanged |
                   IWindow::immediateUpdate, 0);
```

```
static void
    setDefaultCell( const ISize& widthAndHeight );
```

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

setRowHeight

Defines the smallest height that the row will have and indicates whether the row is expandable. 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 multicell canvas contains. IMultiCellCanvas determines the maximum number of rows and columns from `addToCell` (p. 474), `setColumnWidth` (p. 480), and `setRowHeight`.

Note: If a multicell canvas is already shown, this function does not cause it to update the canvas itself immediately. You must call `IWindow::refresh` (Vol. II) to update the appearance of the canvas.

IMultiCellCanvas

```
virtual IMultiCellCanvas&
    setRowHeight( unsigned long row,
                  unsigned long heightInPixels,
                  Boolean expandable = false );
```

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

row Identifier for the row. Rows in a multicell canvas are numbered beginning with 1.

heightInPixels Minimum height in pixels.

expandable If set to true, windows in the row are expanded vertically when the canvas is larger than its minimum size. The default is false. Optional.

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an IMultiCellCanvas object, you can use IMultiCellCanvas, ICanvas, and IWindow member functions to query and change individual styles.

convertToGUIStyle

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

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

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

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 484) unless you have changed the style using setDefaultStyle (p. 482).

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

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

setDefaultStyle

Sets the default style for all subsequent multicell canvases.

IMultiCellCanvas

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

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

style A combination of IMultiCellCanvas::Style (p. 484), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects.

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Use layout members to determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

IMultiCellCanvas

layout Computes the child windows' positions and sizes based on the following:

- The contents of the cells
- The minimum size of rows and columns
- The expandability of rows and columns

virtual IMultiCellCanvas&
 layout();

Win
Y

PM
Y

Motif
Y

Inherited Protected Functions

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

- ICanvas::Style (p. 35)
- IWindow::Style (Vol. II)

Once you have constructed an IMultiCellCanvas object, you can use IMultiCellCanvas, ICanvas, and IWindow member functions to query and change individual styles.

classDefaultStyle

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

static const Style
 classDefaultStyle;

Win
Y

PM
Y

Motif
Y

IMultiCellCanvas

dragLines Causes the multicell canvas to have draggable grid lines between the rows and columns of the canvas.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
dragLines;	Y	Y	Y

gridLines Causes the multicell canvas to have grid lines between the rows and columns of the canvas.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
gridLines;	Y	Y	Y

Inherited Public Data

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IMultiCellCanvas

Nested Classes

IMultiCellCanvas contains the following nested classes:

IMultiCellCanvas::Style (see page 487)



IMultiCellCanvas::Style

Derivation IBase
 IBitFlag
 IMultiCellCanvas::Style

Inherited By None.

Header File imcelcv.hpp

The nested class IMultiCellCanvas::Style provides a set of valid multicell canvas styles for IMultiCellCanvas::defaultStyle (p. 482), IMultiCellCanvas::setDefaultStyle (p. 482), and the constructor of the IMultiCellCanvas (p. 470) class. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
 IWindow::Style (Vol. II)

Once you have constructed an IMultiCellCanvas object, you can use IMultiCellCanvas, ICanvas, and IWindow member functions to query and change individual styles.



dragLines and gridLines are not supported in the AIX release.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

IMultiCellCanvas::Style

Inherited Protected Data

IBase		
recoverable	unrecoverable	



INotebook

Derivation

```

IBase
  IVBase
    INotifier
      IWindow
        IControl
          INotebook
  
```

Inherited By None.

Header File inotbk.hpp

Members				
Member	Page	Member	Page	
Constructor	496	minorTabForegroundColor	493	
addFirstPage	499	minorTabForegroundColorId	517	
addLastPage	499	nextPage	503	
addPageAfter	500	notebookSize	505	
addPageBefore	501	orientation	498	
allTabsVisible	518	orientationId	517	
areAllTabsVisible	511	pageBackgroundColor	494	
backgroundColor	493	pageBackgroundColorId	517	
backPagesBottomLeft	518	pageSettings	502	
backPagesBottomRight	518	pageSize	505	
backPagesTopLeft	518	pagesToEnd	506	
backPagesTopRight	518	pagesToMajorTab	506	
binding	492	pagesToMinorTab	506	
calcMinimumSize	515	pmCompatible	520	
classDefaultStyle	518	polygonTabs	520	
convertToGUIStyle	510	previousPage	503	
defaultStyle	510	refreshTabs	511	
enableNotification	498	registerCallbacks	515	
firstPage	502	removeAllPages	504	
handleDrawTabs	519	removePage	504	
hiliteBackgroundColor	493	removeTabSection	504	
isDrawTabsEnabled	511	resetMajorTabBackgroundColor	494	
isEmpty	505	resetMajorTabForegroundColor	494	
isPMCompatible	498	resetMinorTabBackgroundColor	494	
lastPage	502	resetMinorTabForegroundColor	494	
majorTabBackgroundColor	493	resetPageBackgroundColor	495	
majorTabBackgroundColorId	516	roundedTabs	520	
majorTabForegroundColor	493	setBackgroundColor	495	
majorTabForegroundColorId	516	setBinding	492	
majorTabsBottom	519	setDefaultStyle	510	
majorTabsLeft	519	setForegroundColor	495	
majorTabsRight	519	setMajorTabBackgroundColor	495	
majorTabsTop	520	setMajorTabForegroundColor	495	
minorTabBackgroundColor	493	setMajorTabSize	511	
minorTabBackgroundColorId	516	setMinorTabBackgroundColor	496	

INotebook

Member	Page	Member	Page
setMinorTabForegroundColor	496	statusTextAlignment	509
setMinorTabSize	512	statusTextCenter	520
setOrientation	499	statusTextLeft	521
setPageBackgroundColor	496	statusTextRight	521
setPageButtonSize	501	tabShape	514
setStatusText	508	tabTextAlignment	514
setStatusTextAlignment	509	tabTextCenter	521
setTabBitmap	512	tabTextLeft	521
setTabShape	513	tabTextRight	521
setTabText	513	topPage	503
setTabTextAlignment	514	totalPages	507
setUserData	492	turnToPage	503
setWindow	507	unregisterCallbacks	515
solidBinding	520	window	508
spiralBinding	520	~INotebook	497
squareTabs	520		

The INotebook class is a wrapper for the notebook control. The notebook is a control that is useful in organizing related information on individual pages so that a user can quickly find and display the information. Use an object of this class to create and manage a notebook control.

A notebook control consists of the following items, and this class provides member functions that support each of these items:

- A notebook client area
- A binding
- Page buttons
- Optional tabs
- An optional status area

When working with objects of this class, you can also use the following nested classes:

- INotebook::PageSettings (p. 529), whose objects define the characteristics of a page when it is added to a notebook
- INotebook::Cursor (p. 525), whose objects iterate through the pages of the notebook



The INotebook class supports both the CUA '91 notebook control and the Windows tab control. The CUA '91 notebook control is a ported version of the OS/2 Presentation Manager notebook control. It has the same look-and-feel as the OS/2 Presentation Manager notebook control. The Windows tab control is a control that is native to the Windows environment.

INotebook

Use the style, `INotebook::pmCompatible` (p. 520), to select the CUA '91 notebook control. The Windows tab control is the default selection.



The `INotebook`'s implementation of the Windows tab control only supports major tab pages. However, it will convert all minor and non-tab pages in your existing application to major tab pages. The placement of the major tabs is at the top of the control and their shape is rounded. Tab orientation and shape settings are ignored. The Windows tab control only supports rounded major tabs with their orientation limited to the top of the control. Tab text alignment for the Windows tab control is center. Tab text alignment settings are ignored.

The Windows tab control automatically sizes the tabs, unless you specify a size via `INotebook::setMajorTabSize` (p. 511).

The Windows tab control allows both a bitmap and text on the same tab. The capability to set both a bitmap and text on the same tab is supported by the `INotebook` class.

The Windows tab control supports multiple rows of tabs that keep all of the tabs visible to the user. Specify the notebook style, `INotebook::allTabsVisible` (p. 518), during `INotebook` construction to make all of the tabs visible.

The Windows tab control supports owner drawing of the tabs. However, you must specify the notebook style, `INotebook::handleDrawTabs` (p. 519), during `INotebook` construction to enable the owner draw support. The owner-drawn tabs must all be the same width.

The Windows tab control contains no binding or back pages area. The `INotebook`'s implementation of this control ignores the binding and back pages settings.

The Windows tab control contains no status text line. The `INotebook`'s implementation of this control ignores the status text and text alignment settings.

The Windows tab control contains no page buttons. The `INotebook`'s implementation of this control ignores the page button settings.

The Windows tab control does not provide color support APIs. All tab control colors are based upon the default system colors. The `INotebook`'s implementation of this control ignores the color settings.

Public Functions

INotebook

Application Data

Use these members to set the application data associated with a notebook page.

setUserData Sets the application data into the specified page's reserved storage. Each page has a 4-byte reserved storage area. This area is available for information required by your application.

```
virtual INotebook&
    setUserData( const IPageHandle& referencePage,
                unsigned long userData );
```

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

referencePage Reference to the handle of a notebook page.

userData Unsigned long value that represents the application data to be set into the specified page's storage.

Exceptions		
IAccessError	The user data was not set for the specified page. The page identifier (ID) of the notebook page may be invalid.	

Binding Support

Use these members to query and set the notebook's binding.

binding Returns the type of binding used for the notebook. The return value is a Binding (p. 522) enumerator.

```
virtual Binding
    binding() const;
```

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

setBinding Sets the type of binding for the notebook.

```
virtual INotebook&
    setBinding( Binding binding );
```

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

binding Binding (p. 522) enumeration that specifies the binding type.

Exceptions		
InvalidParameter	The binding style was not set. The <i>binding</i> parameter is invalid.	

Colors

Use these members to set and query the color of different areas of the notebook.

INotebook

backgroundColor

Returns the background color of the notebook or the default if you have not set it.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backgroundColor() const;	<i>I</i>	<i>Y</i>	<i>Y</i>



Returns the default background color of the notebook.

hiliteBackgroundColor

Returns the notebook's highlight background color or the default color value if no highlight background color for the notebook has been set.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hiliteBackgroundColor() const;	<i>I</i>	<i>Y</i>	<i>I</i>



Returns the notebook's default highlight background color.

majorTabBackgroundColor

Returns the major tab background color of the notebook or the default if you have not set it.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTabBackgroundColor() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

majorTabForegroundColor

Returns the major tab foreground color of the notebook or the default if you have not set it.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTabForegroundColor() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

minorTabBackgroundColor

Returns the minor tab background color of the notebook or the default if you have not set it.

virtual IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minorTabBackgroundColor() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

minorTabForegroundColor

Returns the minor tab foreground color of the notebook or the default if you have not set it.

INotebook

```
virtual IColor  
    minorTabForegroundColor() const;
```

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

pageBackgroundColor

Returns the page background color of the notebook or the default if you have not set it.

```
virtual IColor  
    pageBackgroundColor() const;
```

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

resetMajorTabBackgroundColor

Resets the major tab background color by changing to the major tab background color that was previously set.

```
virtual INotebook&  
    resetMajorTabBackgroundColor();
```

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

resetMajorTabForegroundColor

Resets the major tab foreground color by changing to the major tab foreground color that was previously set.

```
virtual INotebook&  
    resetMajorTabForegroundColor();
```

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

resetMinorTabBackgroundColor

Resets the minor tab background color by changing to the minor tab background color that was previously set.

```
virtual INotebook&  
    resetMinorTabBackgroundColor();
```

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

resetMinorTabForegroundColor

Resets the minor tab foreground color by changing to the minor tab foreground color that was previously set.

```
virtual INotebook&  
    resetMinorTabForegroundColor();
```

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

INotebook

resetPageBackgroundColor

Resets the page background color by changing to the page background color that was previously set.

<code>virtual INotebook& resetPageBackgroundColor();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

setBackgroundColor

Sets the notebook's background color.

<code>virtual INotebook& setBackgroundColor(const IColor& color);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>N</i>	<i>N</i>	<i>Y</i>

color Reference to a color object that defines the new background color.

setForegroundColor

Sets the notebook's foreground color.

<code>virtual INotebook& setForegroundColor(const IColor& color);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>N</i>	<i>N</i>	<i>Y</i>

color Reference to a color object that defines the new foreground color.

setMajorTabBackgroundColor

Sets the major tab background color.

<code>virtual INotebook& setMajorTabBackgroundColor(const IColor& color);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

color Reference to a color object that defines the major tab background color.

setMajorTabForegroundColor

Sets the major tab foreground color.

<code>virtual INotebook& setMajorTabForegroundColor(const IColor& color);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

color Reference to a color object that defines the major tab foreground color.

INotebook

setMinorTabBackgroundColor

Sets the minor tab background color.

<code>virtual INotebook& setMinorTabBackgroundColor(const IColor& color);</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

color Reference to a color object that defines the minor tab background color.

setMinorTabForegroundColor

Sets the minor tab foreground color.

<code>virtual INotebook& setMinorTabForegroundColor(const IColor& color);</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

color Reference to a color object that defines the minor tab foreground color.

setPageBackgroundColor

Sets the page background color.

<code>virtual INotebook& setPageBackgroundColor(const IColor& color);</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

color Reference to a color object that defines the page background color.

Constructors

You can construct and destruct objects of this class. This class supports the same three standard constructors that all of the control classes in the User Interface Class Library support.

Note: Typically, you create notebooks as clients of frame windows. Use `IFrameWindow::setClient` (Vol. II) to make the notebook the client of a frame window.

INotebook

1 <code>INotebook(unsigned long windowId, IWindow* parent, IWindow* owner, const IRectangle& initial = IRectangle (), const Style& style = defaultStyle ());</code>	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

windowId Unsigned long value that represents the notebook control identifier (ID).

parent Pointer to the parent window object.

INotebook

<i>owner</i>	Pointer to the owner window object.
<i>initial</i>	Reference to a rectangle object. It specifies the initial position and size of the notebook. The default size and position are determined by the default constructor for IRectangle (Vol. I). Optional.
<i>style</i>	Use the styles provided by INotebook (p. 516) Styles to specify the control's styles. Optional.

Use this version of the constructor to create a new notebook control window and a corresponding INotebook object.

Exceptions	
InvalidParameter	The notebook object was not created. The creation failed because the pointer to the parent window object cannot be 0.

2 INotebook(unsigned long windowId, Win PM Motif
IWindow* parentAndOwner); Y Y N

windowId Unsigned long value that represents the notebook control identifier (ID).

parentAndOwner

Pointer to the parent and owner window object.

Use this version of the constructor if you want to wrapper a notebook control that is loaded from a dialog resource.

Note: Loading a notebook control from a dialog resource limits the capability of the notebook because page windows cannot be specified within a dialog template.

Exceptions	
InvalidParameter	The notebook object was not created. The creation failed because the pointer to the parent and owner window object cannot be 0.

3 INotebook(const IWindowHandle& handle); Win PM Motif
Y Y N

handle Reference to a window handle of an existing notebook control.

Use this version of the constructor to wrapper an existing notebook control window.

~INotebook

This destructor destroys the notebook control window only if the User Interface Class Library created it.

INotebook

<pre>virtual ~INotebook();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Native Control Support

Use these members for native control support.

isPMCompatible

Returns true if the INotebook object represents a CUA '91 notebook control. False is returned if the INotebook object represents a Windows tab control.

<pre>Boolean isPMCompatible() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

PM Always returns true.

Notification Members

Use these INotificationId (Vol. I) strings for all notifications that INotebook provides to its observers.

enableNotification

Enables the notebook to send notifications to any observer objects.

<pre>virtual IWindow& enableNotification(Boolean enable = true);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

enable Boolean value that specifies if the observer notification is enabled. The default is true.

Orientation

Use these members to query and set the orientation of the corner of the back pages of a notebook in relation to the location of the major tabs. The back pages are the recessed edges that give the notebook a three-dimensional effect.

orientation Returns the current orientation of the notebook, which is the location of the major tabs in relation to the back pages' corner. The return value is an Orientation (p. 522) enumerator.

<pre>virtual Orientation orientation() const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

setOrientation

Sets the orientation of the back pages' corner in relation to the location of the major tabs.

virtual INotebook& setOrientation(Orientation orientation);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

orientation Orientation (p. 522) enumeration that specifies the orientation of the back pages' corner in relation to the location of the major tabs.

Exceptions	
InvalidParameter	The orientation style was not set. The <i>orientation</i> parameter is invalid.

Page Addition

Use these members to add a page to a notebook.

addFirstPage Adds a page as the first page in the notebook using the specified page settings. If *pageWindow* is 0, a page with the indicated settings, but no application page window, is added to the notebook.

Note: When adding an IFrameWindow (Vol. II) as a notebook page, create an IFrameWindow with the notebook as its parent.

virtual IPageHandle addFirstPage(const PageSettings& pageInfo, IWindow* pageWindow = 0);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

pageWindow
Pointer to the window object to use as the application page window. The default is 0.

addLastPage Adds a page as the last page in the notebook using the specified page settings. If *pageWindow* is 0, a page with the indicated settings, but no application page window, is added to the notebook.

Note: When adding an IFrameWindow (Vol. II) as a notebook page, create an IFrameWindow with the notebook as its parent.

virtual IPageHandle addLastPage(const PageSettings& pageInfo, IWindow* pageWindow = 0);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

INotebook

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

pageWindow Pointer to the window object to use as the application page window. The default is 0.

addPageAfter

Adds a page to the notebook following the specified page using the specified page settings. If *pageWindow* is 0, a page with the indicated settings, but no application page window, is added to the notebook.

Note: When adding an IFrameWindow (Vol. II) as a notebook page, create an IFrameWindow with the notebook as its parent.

1	<pre>virtual IPageHandle addPageAfter(const PageSettings& pageInfo, const Cursor& cursor, IWindow* pageWindow = 0);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

cursor Reference to a cursor object that points to a notebook page. The new page is added after this page.

pageWindow Pointer to the window object to use as the application page window. The default is 0.

2	<pre>virtual IPageHandle addPageAfter(const PageSettings& pageInfo, const IPageHandle& referencePage, IWindow* pageWindow = 0);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

referencePage Reference to the handle of a notebook page. The new page is added after this page.

pageWindow Pointer to the window object to use as the application page window. The default is 0.

addPageBefore

Adds a page to the notebook before the specified page using the specified page settings. If *pageWindow* is 0, a page with the indicated settings, but no application page window, is added to the notebook.

Note: When adding an IFrameWindow (Vol. II) as a notebook page, create an IFrameWindow with the notebook as its parent.

1	<pre>virtual IPageHandle addPageBefore(const PageSettings& pageInfo, const Cursor& cursor, IWindow* pageWindow = 0);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

cursor Reference to a cursor object that points to a notebook page. The new page is added before this page.

pageWindow Pointer to the window object to use as the application page window. The default is 0.

2	<pre>virtual IPageHandle addPageBefore(const PageSettings& pageInfo, const IPageHandle& referencePage, IWindow* pageWindow = 0);</pre>	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td><i>Y</i></td> <td><i>Y</i></td> <td><i>Y</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

pageInfo Reference to an INotebook::PageSettings (p. 529) object. Use this object to specify the new page's settings.

referencePage Reference to the handle of a notebook page. The new page is added before this page.

pageWindow Pointer to the window object to use as the application page window. The default is 0.

Page Button

Use these members to set the page button's size.

setPageButtonSize

Sets the size in pixels of the page (arrow) buttons that are used to turn the notebook's pages.

INotebook


```
virtual INotebook&
    setPageButtonSize( const ISize& sizePageButton );
```

Win
Y

PM
Y

Motif
I

sizePageButton
Reference to a size object that represents the new size for page buttons.

 This function is ignored in INotebook's implementation of the Windows tab control.

Exceptions	
IAccessError	The page button size was not set. The page button dimensions may be invalid.

Page Information

Use these members to obtain information on a notebook page.

pageSettings Returns information about the specified page.

1

```
PageSettings
    pageSettings( const IPageHandle& page ) const;
```

Win
Y

PM
Y

Motif
Y

page Reference to the handle of the notebook page to obtain information for.

2

```
PageSettings
    pageSettings( const Cursor& cursor ) const;
```

Win
Y

PM
Y

Motif
Y

cursor Reference to a cursor object that points to the notebook page to obtain information for.

Page Manipulation

Use these members to manipulate pages within a notebook.

firstPage Returns an IPageHandle (p. 549) object that references the first page in the notebook.

```
virtual IPageHandle
    firstPage() const;
```

Win
Y

PM
Y

Motif
Y

lastPage Returns an IPageHandle (p. 549) object that references the last page in the notebook.

```
virtual IPageHandle
    lastPage() const;
```

Win
Y

PM
Y

Motif
Y

INotebook

nextPage Returns an IPageHandle (p. 549) object that references the page following the specified page.

virtual IPageHandle nextPage(const IPageHandle& referencePage) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

referencePage Reference to the handle of the notebook page that precedes the page you want to access.

previousPage Returns an IPageHandle (p. 549) object that references the page before the specified page.

virtual IPageHandle previousPage(const IPageHandle& referencePage) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

referencePage Reference to the handle of the notebook page that follows the page you want to access.

topPage Returns an IPageHandle (p. 549) object that references the current top page of the notebook.

virtual IPageHandle topPage() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

turnToPage Moves a specified page to the top of the notebook.

1 virtual INotebook& turnToPage(const Cursor& cursor);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
--	-----------------	----------------	-------------------

cursor Reference to a cursor object that points to the notebook page to move to the top of the notebook.

2 virtual INotebook& turnToPage(const IPageHandle& page);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
---	-----------------	----------------	-------------------

page Reference to the handle of the notebook page to move to the top of the notebook.

INotebook

Exceptions	
IAccessError	The specified page window was not moved to the top of the notebook. The page identifier (ID) of the specified page may be invalid.

Page Removal

Use these members to remove a page or pages from a notebook.

removeAllPages

Removes all the pages in the notebook.

<code>virtual INotebook& removeAllPages();</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	------------------------	-----------------------	--------------------------

removePage Removes the specified page from the notebook.

1	<code>virtual INotebook& removePage(const IPageHandle& page);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	------------------------	-----------------------	--------------------------

page Reference to the handle of the notebook page to remove.

2	<code>virtual INotebook& removePage(const Cursor& cursor);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	---	------------------------	-----------------------	--------------------------

cursor Reference to a cursor object that points to the notebook page to remove.

removeTabSection

If the specified page is a major tab page, it removes the specified page and all subsequent pages up to the next major tab page. If the specified page is a minor tab page, it removes the specified page and all subsequent pages up to the next page that has a tab.

1	<code>virtual INotebook& removeTabSection(const IPageHandle& page);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	------------------------	-----------------------	--------------------------

page Reference to the handle of the notebook page. This page is at the beginning of the section to remove.

Win This function is ignored in INotebook's implementation of the Windows tab control.

INotebook

Exceptions	
InvalidRequest	The notebook tab section was not removed. The removal failed because the specified notebook page is not associated with a major or a minor tab.

2	virtual INotebook& removeTabSection(const Cursor& cursor);	Win Y	PM Y	Motif Y
----------	---	-----------------	----------------	-------------------

cursor Reference to a cursor object that points to the notebook page. This page is at the beginning of the section to remove.

Win This function is ignored in INotebook's implementation of the Windows tab control.

Page Support

Use these members to support page counting and to determine size information for the notebook.

isEmpty Queries whether the notebook has any pages and returns true if it is empty.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isEmpty() const;	γ	γ	γ

notebookSize

Returns the size that the notebook must be in order to contain the specified page. The notebook size is calculated based on the size of the page window, or its minimum size if it has no size.

virtual ISize	Win	PM	Motif
notebookSize(const IPageHandle& page) const;	Y	Y	Y

page Reference to the handle of the notebook page that the size is based on.

Exceptions	
IAccessError	The notebook size was not returned. The coordinates of the application page window may be invalid.

pageSize Returns the size of the notebook's pages. All notebook pages have the same size. Application windows associated with notebook pages can use this information to determine their size.

virtual ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pageSize() const;	Y	Y	Y

INotebook

Exceptions	
IAccessError	The page's size was not returned. The coordinates of the notebook window may be invalid.

pagesToEnd Returns the number of pages in the notebook from a specified notebook page to the end of the notebook. This value includes the specified page.

1 virtual unsigned long Win PM Motif
pagesToEnd(const IPageHandle& page) const; Y Y Y

page Reference to the handle of the notebook page.

2 virtual unsigned long Win PM Motif
pagesToEnd(const Cursor& cursor) const; Y Y Y

cursor Reference to a cursor object that points to the notebook page.

pagesToMajorTab

Returns the number of pages in the notebook between a specified notebook page and the next page that has a major tab. This value does not include the specified page.

1 virtual unsigned long Win PM Motif
pagesToMajorTab(const IPageHandle& page) const; Y Y Y

page Reference to the handle of the notebook page.

Win Always returns 1 for INotebook's implementation of the Windows tab control because every page is a major tab page.

2 virtual unsigned long Win PM Motif
pagesToMajorTab(const Cursor& cursor) const; Y Y Y

cursor Reference to a cursor object that points to the notebook page.

Win Always returns 1 for INotebook's implementation of the Windows tab control because every page is a major tab page.

pagesToMinorTab

Returns the number of pages in the notebook between a specified notebook page and the next page that has a minor tab. This value does not include the specified page.

INotebook

1 virtual unsigned long
pagesToMinorTab(const Cursor& cursor) const; Win PM Motif
Y Y Y

cursor Reference to a cursor object that points to the notebook page.

Win Always returns 0 for INotebook's implementation of the Windows tab control because minor tabs are not supported.

2 virtual unsigned long
pagesToMinorTab(const IPageHandle& page) const; Win PM Motif
Y Y Y

page Reference to the handle of the notebook page.

Win Always returns 0 for INotebook's implementation of the Windows tab control because minor tabs are not supported.

totalPages Returns the total number of pages in the notebook.

virtual unsigned long
totalPages() const; Win PM Motif
Y Y Y

Page Window Association

Use these members to query and set the application page window that is associated with the specified notebook page.

setWindow Associates the specified note page with the application page window.

Note: If you use a page selection handler to delay creating an application page window to associate with a notebook page, you can call this function in the selection handler routine.

1 virtual INotebook&
setWindow(const IPageHandle& referencePage,
IWindow* pageWindow = 0); Win PM Motif
Y Y Y

referencePage
Reference to the handle of the notebook page.

pageWindow
Pointer to the window object to use as the application page window. The default is 0.

Exceptions	
IAccessError	The specified page was not associated with the window. The page identifier (ID) of the notebook page may be invalid.

INotebook

2 virtual INotebook&
setWindow(const Cursor& cursor, Win PM Motif
IWindow* pageWindow = 0); Y Y Y

cursor Reference to a cursor object that points to the notebook page.

pageWindow

Pointer to the window object to use as the application page window. The default is 0.

window Returns the application page window associated with the specified notebook page.

1 virtual IWindow* Win PM Motif
window(const IPageHandle& page) const; Y Y Y

page Reference to the handle of the notebook page.

Exceptions	
IAccessError	The window associated with the specified notebook page was not returned. The page identifier (ID) of the specified notebook page may be invalid.

2 virtual IWindow* Win PM Motif
window(const Cursor& cursor) const; Y Y Y

cursor Reference to a cursor object that points to the notebook page.

Status Text

Use these members to query and set the status text alignment and to set the status text.

setStatusText

Sets the status text for the specified page of the notebook.

1 virtual INotebook& Win PM Motif
setStatusText(const IPageHandle& referencePage, Y Y Y
const IResourceId& resourceId);

referencePage

Reference to the handle of the specified notebook page.

resourceId Reference to a resource identifier object that identifies the text to use on the status line of the specified notebook page.

INotebook

2 `virtual INotebook&
setStatusText(const IPageHandle& referencePage,
const char* statusText);` **Win** **PM** **Motif**
Y Y Y

referencePage

Reference to the handle of the specified notebook page.

statusText Pointer to the text to use on the status line of the specified notebook page.

Exceptions	
IAccessError	The status text was not set for the specified page. The page identifier (ID) of the notebook page may be invalid.
InvalidRequest	The page does not have a status line to display the text. You must specify the <code>INotebook::PageSettings::statusTextOn</code> attribute when adding the page to the notebook.

setStatusTextAlignment

Sets the alignment of the text in the notebook's status line.

`virtual INotebook&
setStatusTextAlignment(TextAlignment alignment);` **Win** **PM** **Motif**
Y Y Y

alignment `TextAlignment` (p. 524) enumeration that specifies the alignment of the status text.

Exceptions	
InvalidParameter	The alignment style of the status text line was not set. The <i>alignment</i> parameter is invalid.

statusTextAlignment

Returns the alignment of the text in the notebook's status line. The return value is a `TextAlignment` (p. 524) enumerator.

`virtual TextAlignment
statusTextAlignment() const;` **Win** **PM** **Motif**
Y Y Y

Styles

Use these members to set and query the various styles that are associated with a notebook. You can use these styles with the styles defined by the following nested classes:

`IWindow::Styles` (Vol. II)
`IControl::Styles` (Vol. II)

INotebook

The combination of styles you specify for the back pages (the recessed edges that give the notebook a three-dimensional effect) and the major tabs determines how the pages in the notebook look and behave when the user iterates through the notebook's pages.

An object of this nested class is provided when you create a notebook. A customizable default is used if you do not specify any styles. Once the notebook object is created, you can use INotebook, IWindow, and IControl member functions to set or query the object's style.

The declaration of the INotebook::Style nested class is generated by the INESTEDBITFLAGCLASSDEF2 macro.

convertToGUIStyle

Converts style bits into the style value that can be processed by the graphical user interface (GUI). The default action is to return the base GUI style for the platform. Extended styles, those that either the User Interface Class Library has defined or you have defined, can be returned by setting the *extendedOnly* parameter to true.

virtual unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
convertToGUIStyle(const IBitFlag& style,		Y	Y	N
Boolean extendedOnly = false) const;				

style Reference to a bit flag object that defines the graphical user interface (GUI) style.

extendedOnly
Boolean value that specifies to return only the extended styles.

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 518) unless you have changed the style using setDefaultStyle (p. 510).

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

setDefaultStyle

Sets the default style for all subsequent notebooks.

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

style Reference to a style object that defines the default style. Use the styles provided by INotebook (p. 516) Styles to specify the default style.

Tab Support

Use these members to provide the following tab support:

- Query and set the tab shape
- Query and set the tab text alignment
- Set the size of the major notebook tabs
- Set the size of the minor notebook tabs
- Refresh (repaint) all of the notebook's tabs
- Set the tab's text
- Set the tab's bitmap

areAllTabsVisible

Returns true if the all tabs visible style, `INotebook::allTabsVisible` (p. 518), is specified. False is always returned if the style `INotebook::pmCompatible` (p. 520) is specified.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>areAllTabsVisible() const;</code>	<i>Y</i>	<i>N</i>	<i>N</i>

isDrawTabsEnabled

Returns true if the owner draw tabs style, `INotebook::handleDrawTabs` (p. 519), is specified. False is always returned if the style `INotebook::pmCompatible` (p. 520) is specified.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>isDrawTabsEnabled() const;</code>	<i>Y</i>	<i>N</i>	<i>N</i>

refreshTabs Causes all of the notebook's visible tabs to be repainted.

<code>virtual INotebook& refreshTabs();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Exceptions	
<code>IAccessError</code>	The tabs were not repainted. An error occurred within the notebook control.

setMajorTabSize

Sets the size of the notebook's major tabs in pixels.

<code>virtual INotebook& setMajorTabSize(const ISize& sizeMajorTab);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>I</i>

INotebook

sizeMajorTab

Reference to a size object that represents the new size for the major tabs.

Exceptions	
IAccessError	The major tab size was not set. The tab dimensions may be invalid.

setMinorTabSize

Sets the size of the notebook's minor tabs in pixels.

```
virtual INotebook&
    setMinorTabSize( const ISize& sizeMinorTab );
```

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

sizeMinorTab

Reference to a size object that represents the new size for minor tabs.



This function is ignored in INotebook's implementation of the Windows tab control.

Exceptions	
IAccessError	The minor tab size was not set. The tab dimensions may be invalid.

setTabBitmap

Sets the tab bitmap for the specified page of the notebook.

1

```
virtual INotebook&
    setTabBitmap( const IPageHandle& referencePage,
                  const IBitmapHandle& bitmap );
```

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

referencePage

Reference to the handle of the specified notebook page.

bitmap

Reference to the handle of the bitmap to use on the tab of the specified notebook page.

Exceptions	
IAccessError	The tab bitmap was not set for the specified page. The page identifier (ID) of the notebook page may be invalid or the page may be missing the major or minor tab attribute.

2

```
virtual INotebook&
    setTabBitmap( const IPageHandle& referencePage,
                  const IResourceId& resourceId );
```

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

referencePage

Reference to the handle of the specified notebook page.

INotebook

resourceId Reference to a resource identifier object that identifies the bitmap to use on the tab of the specified notebook page.

setTabShape Sets the shape of the notebook's tabs.

```
virtual INotebook&
    setTabShape( TabShape tabShape );
```

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

tabShape TabShape (p. 523) enumeration that specifies the shape of the tab.

Exceptions	
InvalidParameter	The tab shape style was not set. The <i>tabShape</i> parameter is invalid.

setTabText Sets the tab text for the specified page of the notebook.

Note: Multiple lines of text on the tabs are not supported. You must use the tab owner draw support to implement multiline text.

1

```
virtual INotebook&
    setTabText( const IPageHandle& referencePage,
                const IResourceId& resourceId );
```

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

referencePage

Reference to the handle of the specified notebook page.

resourceId Reference to a resource identifier object that identifies the text to use on the tab of the specified notebook page.

2

```
virtual INotebook&
    setTabText( const IPageHandle& referencePage,
                const char* tabText );
```

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

referencePage

Reference to the handle of the specified notebook page.

tabText Pointer to the text to use on the tab of the specified notebook page.

Exceptions	
IAccessError	The tab text was not set for the specified page. The page identifier (ID) of the notebook page may be invalid or the page may be missing the major or minor tab attribute.

INotebook

setTabTextAlignment

Sets the alignment of the text in the notebook's tabs.

```
virtual INotebook&
    setTabTextAlignment( TextAlignment alignment );
```

Win
Y

PM
Y

Motif
Y

alignment TextAlignment (p. 524) enumeration that specifies the alignment of the tab text.

Exceptions	
InvalidParameter	The tab text alignment style was not set. The <i>alignment</i> parameter is invalid.

tabShape

Returns the shape of the notebook's tabs. The return value is a TabShape (p. 523) enumerator.

```
virtual TabShape
    tabShape() const;
```

Win
Y

PM
Y

Motif
I

tabTextAlignment

Returns the alignment of the text in the notebook's tabs. The returned value is a TextAlignment (p. 524) enumerator.

```
virtual TextAlignment
    tabTextAlignment() const;
```

Win
Y

PM
Y

Motif
Y

Inherited Public Functions

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Event-Handling Implementation

Event-handling members add and remove callback functions for events that a notebook might receive.

registerCallbacks

Registers all possible callbacks and X event handlers to an object of this class for events it might receive. IHandler (Vol. II) derived classes later determine which events they process.

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

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
registerCallbacks();	N	N	Y

unregisterCallbacks

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

virtual void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
unregisterCallbacks();	N	N	Y

Layout Support

Use these members to support the layout of a notebook within a canvas.

calcMinimumSize

Returns the minimum recommended size for this notebook. This size is large enough to contain the largest application page window that is currently in the notebook. This function calls notebookSize (p. 505).

INotebook

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

Win
Y

PM
Y

Motif
Y

Exceptions	
IAccessError	The minimum size was not calculated. The coordinates of the notebook or application page window may be invalid.

Inherited Protected Functions

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

Use these INotificationId (Vol. I) strings for all notifications that INotebook provides to its observers.

majorTabBackgroundColorId

Notification identifier provided to observers when the major tab background color of the notebook control changes.

```
static INotificationId const
  majorTabBackgroundColorId;
```

Win
Y

PM
Y

Motif
Y

majorTabForegroundColorId

Notification identifier provided to observers when the major tab foreground color of the notebook control changes.

```
static INotificationId const
  majorTabForegroundColorId;
```

Win
Y

PM
Y

Motif
Y

minorTabBackgroundColorId

Notification identifier provided to observers when the minor tab background color of the notebook control changes.

INotebook

static INotificationId const
minorTabBackgroundColorId;

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

minorTabForegroundColorId

Notification identifier provided to observers when the minor tab foreground color of the notebook control changes.

static INotificationId const
minorTabForegroundColorId;

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

orientationId Notification identifier provided to observers when the orientation style of the notebook control changes. INotebook provides the new orientation value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is one of the predefined Orientation enumeration values defined in this class.

static INotificationId const
orientationId;

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

pageBackgroundColorId

Notification identifier provided to observers when the page background color of the notebook control changes.

static INotificationId const
pageBackgroundColorId;

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

Styles

Use these members to set and query the various styles that are associated with a notebook. You can use these styles with the styles defined by the following nested classes:

IWindow::Styles (Vol. II)
IControl::Styles (Vol. II)

The combination of styles you specify for the back pages (the recessed edges that give the notebook a three-dimensional effect) and the major tabs determines how the pages in the notebook look and behave when the user iterates through the notebook's pages.

An object of this nested class is provided when you create a notebook. A customizable default is used if you do not specify any styles. Once the notebook object is created, you can use INotebook, IWindow, and IControl member functions to set or query the object's style.

The declaration of the INotebook::Style nested class is generated by the INESTEDBITFLAGCLASSDEF2 macro.

INotebook

allTabsVisible

Draws the tabs of the notebook in multiple rows, so all are visible. This style is ignored if the style INotebook::pmCompatible (p. 520) is specified.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
allTabsVisible;	<i>Y</i>	<i>N</i>	<i>N</i>

backPagesBottomLeft

Draws the back pages of the notebook behind the bottom left corner of the notebook.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backPagesBottomLeft;	<i>Y</i>	<i>Y</i>	<i>Y</i>

backPagesBottomRight

Draws the back pages of the notebook behind the bottom right corner of the notebook. This is the default.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backPagesBottomRight;	<i>Y</i>	<i>Y</i>	<i>Y</i>

backPagesTopLeft

Draws the back pages of the notebook behind the top left corner of the notebook.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backPagesTopLeft;	<i>Y</i>	<i>Y</i>	<i>Y</i>

backPagesTopRight

Draws the back pages of the notebook behind the top right corner of the notebook.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
backPagesTopRight;	<i>Y</i>	<i>Y</i>	<i>Y</i>

classDefaultStyle

Specifies the original default style for this class, which is the combination of the following:

- squareTabs
- solidBinding
- statusTextLeft

INotebook

tabTextCenter
backPagesBottomRight
majorTabsRight
IWindow::visible

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

handleDrawTabs

Selects the owner draw tab style, which enables IPageHandler (p. 551) to dispatch the INotebookDrawItemEvent (p. 540). Override IPageHandler::drawTab (p. 553) to process this event.

This style is ignored if the style INotebook::pmCompatible (p. 520) is specified.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
handleDrawTabs;	<i>Y</i>	<i>N</i>	<i>N</i>

majorTabsBottom

Draws the major tabs of the notebook on the bottom of the notebook.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTabsBottom;	<i>Y</i>	<i>Y</i>	<i>Y</i>

majorTabsLeft

Draws the major tabs of the notebook on the left side of the notebook.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTabsLeft;	<i>Y</i>	<i>Y</i>	<i>Y</i>

majorTabsRight

Draws the major tabs of the notebook on the right side of the notebook. This is the default.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTabsRight;	<i>Y</i>	<i>Y</i>	<i>Y</i>

INotebook

majorTabsTop

Draws the major tabs of the notebook on the top of the notebook.

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

pmCompatible

Selects the CUA '91 notebook control. If this style is not set, the Windows tab control is selected.

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

polygonTabs Draws tabs with polygon corners.

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

roundedTabs Draws tabs with rounded corners.

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

solidBinding Draws a solid binding. This is the default.

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

spiralBinding Draws a spiral binding.

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

squareTabs Draws tabs with square corners. This is the default.

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

statusTextCenter

Centers the text in the status line.

I Notebook

```
static const Style
    statusTextCenter;
```

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

statusTextLeft

Left-justifies the text in the status line. This is the default.

```
static const Style
    statusTextLeft;
```

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

statusTextRight

Right-justifies the text in the status line.

```
static const Style
    statusTextRight;
```

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

tabTextCenter

Centers the text in the tabs. This is the default.

```
static const Style
    tabTextCenter;
```

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

tabTextLeft Left-justifies the text in the tabs.

```
static const Style
    tabTextLeft;
```

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

tabTextRight Right-justifies the text in the tabs.

```
static const Style
    tabTextRight;
```

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

Inherited Public Data

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId

INotebook

IWindow		
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

INotebook contains the following nested classes:

INotebook::Cursor (see page 525)

INotebook::Style (see page 539)

INotebook::PageSettings (see page 529)

Binding

```
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.

INotebook

```
Orientation {  
    backpageBottomTabsRight, backpageTopTabsRight,  
    backpageBottomTabsLeft,  backpageTopTabsLeft,  
    backpageRightTabsTop,    backpageLeftTabsTop,  
    backpageRightTabsBottom, backpageLeftTabsBottom  
};
```

Use these enumerators to specify the orientation of the back pages' corner in relation to the location of the major tabs:

backpageBottomTabsRight

Places back pages on the bottom-right corner and the major tabs on the right side of the notebook.

backpageTopTabsRight

Places back pages on the top-right corner and the major tabs on the right side of the notebook.

backpageBottomTabsLeft

Places back pages on the bottom-left corner and the major tabs on the left side of the notebook.

backpageTopTabsLeft

Places back pages on the top-left corner and the major tabs on the left side of the notebook.

backpageRightTabsTop

Places back pages on the top-right corner and the major tabs on the top of the notebook.

backpageLeftTabsTop

Places back pages on the top-left corner and the major tabs on the top of the notebook.

backpageRightTabsBottom

Places back pages on the bottom-right corner and the major tabs on the bottom of the notebook.

backpageLeftTabsBottom

Places back pages on the bottom-left corner and the major tabs on the bottom of the notebook.

TabShape

```
TabShape {  
    square,  
    rounded,  
    polygon  
};
```

Use these enumerators to specify the shape of the notebook's tabs:

INotebook

square

Draws the tabs with square corners.

rounded

Draws the tabs with rounded corners.

polygon

Draws the tabs with polygon corners.

TextAlignment

```
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.



INotebook::Cursor

Derivation IBase
IVBase
INotebook::Cursor

Inherited By None.

Header File inotbk.hpp

Members	Member	Page	Member	Page
	Constructor	525	previous	527
	current	527	setCurrent	527
	Cursor	525	setToFirst	526
	first	527	setToLast	526
	invalidate	526	setToNext	526
	isValid	526	setToPrevious	526
	last	527	~Cursor	526
	next	527		

The nested class INotebook::Cursor defines objects that you can use to iterate through the pages of a notebook. In the same way that you can use a cursor to iterate through the objects in a collection, you can use this cursor object to iterate through a notebook, one page at a time.

An object of this class processes IPageHandle (p. 549) objects, each of which can be thought of as a pointer to a specific page in the notebook.

Public Functions

Constructors

You can construct and destruct objects of this class.

Cursor Create objects of this nested class by specifying a notebook. Use the object to iterate through the pages of the notebook that you specified.

```
Cursor( const INotebook& notebook );
```

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

notebook Reference to the notebook object.

INotebook::Cursor

~Cursor

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

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

Page Iteration

Use these members to iterate through the pages of a notebook.

invalidate Flags this cursor as not valid.

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

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

isValid Queries whether the cursor points to a valid item and returns true if it does.

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

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

setToFirst Sets the cursor to point to the first page of the notebook.

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

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

setToLast Sets the cursor to point to the last page of the notebook.

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

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

setToNext Sets the cursor to point to the next page of the notebook.

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

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

setToPrevious

Sets the cursor to point to the previous page of the notebook.

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

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

Page Retrieval

Use these members to retrieve notebook pages.

current	Returns an IPageHandle (p. 549) object for the notebook page to which the cursor currently points.			
	virtual IPageHandle current() const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
first	Points to the first page in the notebook and returns an IPageHandle (p. 549) object that points to the first notebook page.			
	virtual IPageHandle first();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
last	Points to the last page in the notebook and returns an IPageHandle (p. 549) object that points to the last notebook page.			
	virtual IPageHandle last();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
next	Points to the next page in the notebook and returns an IPageHandle (p. 549) object that points to the next notebook page.			
	virtual IPageHandle next();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
previous	Points to the previous page in the notebook and returns an IPageHandle (p. 549) object that points to the previous notebook page.			
	virtual IPageHandle previous();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y
setCurrent	Sets the cursor to point to the specified notebook page, therefore making it the current page.			
	void setCurrent(const IPageHandle& current);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> Y

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



INotebook::PageSettings

Derivation IBase
IVBase
INotebook::PageSettings

Inherited By None.

Header File inotbk.hpp

Members	Member	Page	Member	Page
	Constructor	531	setStatusText	533
	autoPageSize	535	setTabBitmap	534
	isAutoSize	530	setTabText	534
	isMajorTab	533	setUserData	530
	isMinorTab	533	statusText	533
	isStatusTextOn	532	statusTextOn	535
	majorTab	535	tabBitmap	534
	minorTab	535	tabText	534
	noAttribute	535	userData	530
	operator =	530	~PageSettings	532
	PageSettings	531		

The nested class INotebook::PageSettings identifies information about a page in a notebook. You create objects of this nested class to define characteristics of pages when adding the pages to a notebook. You can use one object of this nested class to create multiple pages that have the same characteristics.

You can also create an object of this nested class to reflect the current state of a page in a notebook.

The INotebook::PageSettings object only identifies the characteristics of a page at one point in time, such as when you are adding the page to a notebook. The object is not updated when the actual page changes. Also, changes to the INotebook::PageSettings object are not reflected in the actual page after it is added to a notebook.



The INotebook's implementation of the Windows tab control only supports major tab pages. However, it will convert all minor and non-tab pages in your existing application to major tab pages.

The Windows tab control contains no status text line. The INotebook's implementation of this control ignores the status text and text alignment settings.

INotebook::PageSettings

Public Functions

Application Data

Use these members to query and set the application data associated with an INotebook::PageSettings object.

Note: Once an INotebook::PageSettings object has been used to insert a page into a notebook, using these members on the object has no effect on the application data within the actual notebook page; therefore, you should use the corresponding application data members located within INotebook. However, you can still use these members to update the application data associated with the INotebook::PageSettings object, and you can use the object to define a new notebook page.

setUserData Sets the application data into the page's reserved storage. Each page has a 4-byte reserved storage area. This area is available for information required by your application.

<code>virtual INotebook::PageSettings& setUserData(unsigned long userData);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

userData Unsigned long value that represents the application data to be set into the page's storage.

userData Returns the application data from the page's reserved storage.

<code>virtual unsigned long userData() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Automatic Sizing

Use these members to determine if the page is automatically sized by the notebook.

isAutoSize Queries whether the page is automatically sized by the notebook and returns true if it is.

<code>Boolean isAutoSize() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Constructors

You can construct and destruct objects of this class.

operator = Assigns an INotebook::PageSettings object to reference an existing page settings object.

INotebook::PageSettings

```
PageSettings&  
operator =( const INotebook::PageSettings& pageSettings );
```

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

pageSettings

A reference to the existing page settings object.

PageSettings You can create objects of this class by specifying various parameters.

1

```
PageSettings();
```

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

You can create an object of this nested class by using this, the default constructor.
The User Interface Class Library defines the default attribute to be
INotebook::PageSettings::noAttribute, and all of the other page settings values are set
to 0.

2

```
PageSettings( const Attribute& attribute );
```

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

attribute Reference to an INoteBook::PageSettings::Attribute object.

You can create an object of this nested class by using one or more page attributes.
All of the other page settings are set to 0.

3

```
PageSettings( const char* tabText,  
              const char* statusText = 0,  
              const Attribute& pageAttribute = noAttribute );
```

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

tabText Pointer to the text to use on the notebook page's tab.

statusText Pointer to the text to use on the status line of the notebook page.

pageAttribute

Reference to an INoteBook::PageSettings::Attribute object. The default is
INotebook::PageSettings::noAttribute.

You can create an object of this nested class by using the tab text, status text, and
one or more attributes. Use this constructor when you do not want to specify a
bitmap for the tab.

4

```
PageSettings( const IBitmapHandle& tabBitmap,  
              const char* statusText = 0,  
              const Attribute& pageAttribute = noAttribute );
```

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

INotebook::PageSettings

tabBitmap Reference to the handle of the bitmap to use on the notebook page's tab.

statusText Pointer to the text to use on the status line of the notebook page.

pageAttribute

Reference to an INotebook::PageSettings::Attribute object. The default is INotebook::PageSettings::noAttribute.

You can construct an object of this nested class by using the tab bitmap, status text, and one or more attributes. Use this constructor when you do not want to specify text for the tab.

```
5 PageSettings( const INotebook::PageSettings& pageSettings );
```

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

pageSettings

Reference to the page settings object to copy.

Use this constructor to copy another INotebook::PageSettings object.

~PageSettings

```
virtual
```

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

```
~PageSettings();
```

Status Text

Use these members to query and set the status text associated with an INotebook::PageSettings object.

Note: Once an INotebook::PageSettings object has been used to insert a page into a notebook, using these members on the object has no effect on the status text within the actual notebook page; therefore, you should use the corresponding status text members located within INotebook. However, you can still use these members to update the status text associated with the INotebook::PageSettings object, and you can use the object to define a new notebook page.

isStatusTextOn

Queries whether a status line is associated with the page and returns true if one is.

```
Boolean
```

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

```
isStatusTextOn() const;
```


INotebook::PageSettings

setStatusText

Sets the text of the status line associated with this page. Calling this function adds the INotebook::PageSettings::statusTextOn (p. 535) attribute to the settings object.

1	<code>virtual INotebook::PageSettings& setStatusText(const IResourceId& resourceId);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

resourceId Reference to a resource identifier object that identifies the text to use on the status line of the notebook page.

2	<code>virtual INotebook::PageSettings& setStatusText(const char* statusText);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

statusText Pointer to the text to use on the status line of the notebook page.

statusText Returns the text of the status line associated with this page.

<code>virtual IString statusText() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Tabs

Use these members to query and set the tab's text or bitmap associated with an INotebook::PageSettings object.

Note: Once an INotebook::PageSettings object has been used to insert a page into a notebook, using these members on the object has no effect on the notebook page's tab; therefore, you should use the corresponding tab members located within INotebook. However, you can still use these members to update the tab text or bitmap associated with the INotebook::PageSettings object, and you can use the object to define a new notebook page.

isMajorTab Queries whether a major tab is associated with the page and returns true if one is.

<code>Boolean isMajorTab() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

isMinorTab Queries whether a minor tab is associated with the page and returns true if one is.

<code>Boolean isMinorTab() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

INotebook::PageSettings

setTabBitmap

Sets the bitmap of the tab associated with this page.

1	<code>virtual INotebook::PageSettings& setTabBitmap(const IResourceId& resourceId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

resourceId Reference to a resource identifier object that identifies the bitmap to use on the notebook page's tab.

2	<code>virtual INotebook::PageSettings& setTabBitmap(const IBitmapHandle& bitmap);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

bitmap Reference to the handle of the bitmap to use on the notebook page's tab.

setTabText

Sets the text of the tab associated with this page.

Note: Multiple lines of text on the tabs are not supported. You must use the tab owner draw support to implement multiline text.

1	<code>virtual INotebook::PageSettings& setTabText(const IResourceId& resourceId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

resourceId Reference to a resource identifier object that identifies the text to use on the notebook page's tab.

2	<code>virtual INotebook::PageSettings& setTabText(const char* tabText);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
----------	--	-------------------------------	------------------------------	---------------------------------

tabText Pointer to the text to use on the notebook page's tab.

tabBitmap

Returns the bitmap of the tab associated with this page.

<code>virtual IBitmapHandle tabBitmap() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

tabText

Returns the text of the tab associated with this page.

<code>virtual IString tabText() const;</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>Y</i>
---	-------------------------------	------------------------------	---------------------------------

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Public Data

Attributes

These members define the various attributes that are associated with a notebook page.

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.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
autoPageSize;	Y	Y	Y

majorTab Specifies that the page is associated with a major tab.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
majorTab;	Y	Y	Y

minorTab Specifies that the page is associated with a minor tab.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
minorTab;	Y	Y	Y

noAttribute Specifies that no attribute applies to the notebook page.

static const Attribute	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noAttribute;	Y	Y	Y

statusTextOn Specifies that the page has a status line. You cannot assign status text to a page without a status line.

INotebook::PageSettings

static const Attribute
statusText0n;

Win
Y

PM
Y

Motif
Y

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

INotebook::PageSettings contains the following nested classes:

INotebook::PageSettings::Attribute (see page 537)



INotebook::PageSettings::Attribute

Derivation IBase
 IBitFlag
 INotebook::PageSettings::Attribute

Inherited By None.

Header File inotbk.hpp

The nested class INotebook::PageSettings::Attribute provides a set of valid notebook page attributes for the member functions of the nested class INotebook::PageSettings (p. 529).

An object of this class is provided when the notebook page is created.

The declaration of the INotebook::PageSettings::Attribute nested class is generated by the INESTEDBITFLAGCLASSDEF0 macro.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

INotebook::PageSettings::Attribute

IBase		
recoverable	unrecoverable	



INotebook::Style

Derivation IBase
 IBitFlag
 INotebook::Style

Inherited By None.

Header File inotbk.hpp

The nested class INotebook::Style provides static members that define the set of valid notebook styles for the class INotebook (p. 489).

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



INotebookDrawItemEvent

INotebookDrawItemEvent



Inherited By None.

Header File inbdievt.hpp

Members	Member	Page	Member	Page
	Constructor	540	page	541
	itemId	541	~INotebookDrawItemEvent	540

The INotebookDrawItemEvent class provides event information for drawing a tab on a notebook page. This draw-item event is generated only if a notebook page has a tab that does not have text or a bitmap attached to it.

Public Functions

Constructors

Use these members to create objects of this class.

INotebookDrawItemEvent

Constructs an object of this class from a reference to an object of the IEvent (Vol. II) class.

INotebookDrawItemEvent(IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>
<i>event</i>	Reference to an event object.		

~INotebookDrawItemEvent

virtual ~INotebookDrawItemEvent();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

INotebookDrawItemEvent

Page Information

Use these members to obtain information about the notebook page.

itemId Returns an identifier for the page whose tab is to be drawn.
INotebookDrawItemEvent::page (p. 541) uses this value to return an IPageHandle (p. 549).

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

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

page Returns a handle to the page whose tab is to be drawn.

```
virtual IPageHandle  
    page() const;
```

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

Inherited Public Functions

IDrawItemEvent		
itemId	itemPresSpaceHandle	itemRect

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

INotebookDrawItemEvent

Inherited Protected Functions

IDrawItemEvent		
ownerItemData		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



INotebookNotifyHandler

Derivation IBase
 IVBase
 IHandler
 IWindowNotifyHandler
 INotebookNotifyHandler

Inherited By None.

Header File inotebnh.hpp

Members	Member	Page
	Constructor	543
	dispatchHandlerEvent	544
	~INotebookNotifyHandler	543

The INotebookNotifyHandler class processes events for all classes of notebooks.

This class is designed to handle events that require the notebook class to generate a notification. If notifications are enabled for this class, a notification is generated and sent to all observers when the proper conditions for the specific notification exist.

Public Functions

Constructors

You can construct and destruct objects of this class.

INotebookNotifyHandler

Provides the default constructor.

INotebookNotifyHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

~INotebookNotifyHandler

virtual ~INotebookNotifyHandler();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

INotebookNotifyHandler

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-processing members evaluate an event to determine if it is appropriate for this handler object to process.

dispatchHandlerEvent

Notifies the notebook observers if any of the following events are received:

- backgroundPageColor event
- backgroundMajorColor event
- backgroundMinorColor event
- foregroundMajorColor event
- foregroundMinorColor event

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

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

Inherited Protected Functions

IWindowNotifyHandler		
dispatchHandlerEvent		

INotebookNotifyHandler

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By IPageHelpEvent
IPageRemoveEvent
IPageSelectEvent

Header File ipageevt.hpp

Members		Member	Page	Member	Page
		Constructor	546	pageHandle	547
		notebook	547	~IPageEvent	547

The IPageEvent class represents a notification for a page-related notebook event. A notebook page handler creates an object of this class when a notebook page is resized. This class is also the base class for other page-related events.

Note: Page events are first dispatched to the notebook and then to the owner window of the notebook.

Public Functions

Constructors

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (p. 553) constructs objects of this class from an object of the class IEvent (Vol. II) or IControlEvent (Vol. II). You can also destruct objects of this class.

IPageEvent

IPageEvent(const IEvent& event);

Win

PM

Motif

Y

Y

Y

event Reference to an event object.

Use this version of the constructor to create a page event object from an event object.

IPageEvent

2	<code>IPageEvent(const IControlEvent& controlEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

controlEvent

Reference to a control event object.

Use this version of the constructor to create a page event object from a control event object.

3	<code>IPageEvent(const IPageEvent& pageEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

pageEvent Reference to a page event object.

Use this version of the constructor, the copy constructor, to create a page event object from another page event object.

~IPageEvent

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IPageEvent();</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

Page Information

Use these members to obtain information about the page that is the subject of the notification event and the notebook.

notebook Returns a pointer to the notebook object that the page belongs to. This page is the target of the notification event.

<code>virtual INotebook*</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>notebook() const;</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

pageHandle Returns the notebook page handle that is the subject of the notification event.

<code>virtual IPageHandle</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>pageHandle() const;</code>		<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Functions

IPageEvent

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IPageHandle

Derivation IBase
IHandle
IPageHandle

Inherited By None.

Header File inotbk.hpp

Members	Member	Page
	Constructor	549
	operator Value	550

The IPageHandle class represents a page in a notebook. The User Interface Class Library only enables notebooks to construct objects of this class. A notebook returns an IPageHandle when the following occurs:

- A page is added to the notebook. The IPageHandle represents the new page.
- A page is queried, such as a call to INotebook::firstPage (p. 502).

You can use an object of this class to identify a page, just as you can identify a page with an INotebook::Cursor (p. 525) object.

Public Functions

Constructors

You can create objects of this class from a page identifier, which defaults to 0.

IPageHandle

```
IPageHandle( Value pageId = 0 );
```

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

pageId IHandle::Value value that represents the page identifier (ID).

Page Handle

Use this operator to return the page handle value.

IPageHandle

operator Value Returns the page handle value.

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

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

Inherited Public Functions

IHandle		
asDebugInfo	asString	asUnsigned

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IHandle		
handle		

IBase		
recoverable	unrecoverable	



IPageHandler

Derivation

- IBase
- IVBase
- IHandler
- IPageHandler

Inherited By None.

Header File ipagehdr.hpp

Members	Member	Page	Member	Page
	Constructor	552	resize	554
	dispatchHandlerEvent	553	select	554
	drawTab	553	selectPending	554
	help	553	~IPageHandler	552
	remove	554		

The IPageHandler class handles the processing of page change events for the INotebook (p. 489) 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 attach it by calling IHandler::handleEventsFor (Vol. II) to identify the notebook or owner window to the page handler.

This class dispatches the following events:

Drawing a tab

INotebookDrawItemEvent (p. 540)

Requesting help for a page

IPageHelpEvent (p. 556)

Removing a page

IPageRemoveEvent (p. 559)

Resizing a page

IPageEvent (p. 546)

Selecting a page

IPageSelectEvent (p. 562)

When the page handler receives one of these events, it creates a corresponding event object and dispatches that object to the appropriate IPageHandler virtual function.

IPageHandler

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 (Vol. II) to dispatch the page event to the notebook’s owner window.
 - If this is the last handler for the owner window, call IWindow::defaultProcedure (Vol. II) to process the page event.

Public Functions

Constructors

You can construct and destruct objects of this class.

IPageHandler Construct objects of this class using this, the default constructor, which does not accept any parameters.

IPageHandler();

Win
Y

PM
Y

Motif
Y

~IPageHandler

virtual
~IPageHandler();

Win
Y

PM
Y

Motif
Y

Inherited Public Functions

IHandler		
asDebugInfo	disable	handleEventsFor
asString	enable	isEnabled

IPageHandler

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Protected Functions

Event Dispatching

Event-dispatching members evaluate an event to determine if it is appropriate for this handler object to process. If it is, this function calls the IPageHandler virtual function used to process the event.

dispatchHandlerEvent

If a page event is received, calls the appropriate IPageHandler virtual function.

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

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

event Reference to an event object.

Event Processing

Event-processing members override these functions in a derived class to process the various notebook page-related events. Use the overridden function to supply the customized processing of a page-related event.

drawTab Implemented by derived classes. It draws the notebook page tab.

```
virtual Boolean
    drawTab( INotebookDrawItemEvent& event );
```

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

event Reference to a notebook event object that provides tab drawing information.

help Implemented by derived classes. It processes a notification that help has been requested for a notebook page.

IPageHandler

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
help(IPageHelpEvent& event);	<i>Y</i>	<i>Y</i>	<i>Y</i>

event Reference to a page help event object.



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 for any other window object.

remove Implemented by derived classes. It processes a page remove notification.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
remove(IPageRemoveEvent& event);	<i>Y</i>	<i>Y</i>	<i>Y</i>

event Reference to a page remove event object.

resize Implemented by derived classes. It processes a notification that the page size changed.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resize(IPageEvent& event);	<i>Y</i>	<i>Y</i>	<i>Y</i>

event Reference to a page event object.

select Implemented by derived classes. It processes the notification that a new page is at the top of the notebook.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
select(IPageSelectEvent& event);	<i>Y</i>	<i>Y</i>	<i>Y</i>

event Reference to a page select event object.

selectPending

Implemented by derived classes. It processes the notification that selection is pending on another page in the notebook.

Use this member to save the state of the outgoing page. To prevent the selection from changing to another page, set the event result to true via IEvent::setResult (Vol. II) and return true on the exit from this member. For example, you might not want to switch from a multicell canvas in which a control has an invalid setting.

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

IPageHandler

event Reference to a page select event object.

Inherited Protected Functions

IHandler		
defaultProcedure	dispatchHandlerEvent	

Inherited Protected Data

IBase		
recoverable	unrecoverable	




Inherited By None.

Header File ipageevt.hpp

Members	Member	Page	Member	Page
	Constructor	556	pageHandle	558
	helpWindow	557	~IPageHelpEvent	557
	notebook	557		

The IPageHelpEvent class represents a notification that help has been requested for a notebook page. A notebook page handler creates and uses an object of this class.

Note: This event is first dispatched to the notebook and then to the owner window of the notebook.


 The AIX release of the User Interface Class Library does not support this class.

Public Functions

Constructors

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (p. 553) constructs objects of this class from an object of the class IEvent (Vol. II), IControlEvent (Vol. II), or IPageEvent (p. 546). You can also destruct objects of this class.

IPageHelpEvent

	IPageHelpEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	I

IPageHelpEvent

event Reference to an event object.

Use this version of the constructor to create a page help event object from an event object.

2	<code>IPageHelpEvent(const IControlEvent& controlEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

controlEvent
Reference to a control event object.

Use this version of the constructor to create a page help event object from a control event object.

3	<code>IPageHelpEvent(const IPageEvent& pageEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

pageEvent Reference to a page event object.

Use this version of the constructor to create a page help event object from a page event object.

~IPageHelpEvent

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IPageHelpEvent();</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

Help Information

Use these members to obtain help information.

helpWindow Returns the help window you can use to display help for the notebook page.

<code>IHelpWindow*</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>helpWindow() const;</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

Page Information

Use these members to obtain information about the page that is the target of the help request and the notebook.

notebook Returns a pointer to the notebook object that the page belongs to.

IPageHelpEvent

```
virtual INotebook*  
    notebook() const;
```

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

pageHandle Returns the handle of the notebook page for which help has been requested.

```
virtual IPageHandle  
    pageHandle() const;
```

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

Inherited Public Functions

IPageEvent		
notebook	pageHandle	

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter 1	setEventType
dispatchingWindow	parameter 2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IPageRemoveEvent

Derivation

```

IBase
IVBase
IEvent
IControlEvent
IPageEvent
IPageRemoveEvent
    
```

Inherited By None.

Header File ipageevt.hpp

Members	Member	Page	Member	Page
	Constructor	559	tabBitmap	560
	notebook	560	~IPageRemoveEvent	560
	pageWindow	560		

The IPageRemoveEvent class represents a notification of the removal of a notebook page. A notebook page handler creates and uses an object of this class.

This event provides functions to return the application page window and bitmap. The application must delete the window and bitmap if they exist.

Note: This event is first dispatched to the notebook and then to the owner window of the notebook.



The AIX release of the User Interface Class Library does not support this class.

Public Functions

Constructors

Although you can construct objects of this class, typically IPageHandler::dispatchHandlerEvent (p. 553) constructs objects of this class from an object of the class IEvent (Vol. II), IControlEvent (Vol. II), or IPageEvent (p. 546). You can also destruct objects of this class.

IPageRemoveEvent

1 IPageRemoveEvent(const IEvent& event);

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

IPageRemoveEvent

event Reference to an event object.

Use this version of the constructor to create a page remove event object from an event object.

2	<code>IPageRemoveEvent(const IControlEvent& controlEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

controlEvent
Reference to a control event object.

Use this version of the constructor to create a page remove event object from a control event object.

3	<code>IPageRemoveEvent(const IPageEvent& pageEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>I</i>

pageEvent Reference to a page event object.

Use this version of the constructor to create a page remove event object from a page event object.

~IPageRemoveEvent

<code>virtual</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IPageRemoveEvent();</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

Page Information

Use these members to obtain information about the deleted page and the notebook.

notebook Returns a pointer to the notebook object that the page belongs to.

<code>virtual INotebook*</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>notebook() const;</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

pageWindow Returns the window or control associated with the page.

<code>IWindow*</code>		<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>pageWindow() const;</code>		<i>Y</i>	<i>Y</i>	<i>I</i>

tabBitmap Returns the handle of the bitmap used by the tab associated with the page.

IPageRemoveEvent

```
IBitmapHandle  
tabBitmap() const;
```

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

Inherited Public Functions

IPageEvent		
notebook	pageHandle	

IControlEvent		
controlId		

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

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File ipageevt.hpp

Members	Member	Page	Member	Page
	Constructor	562	previousSelectedPageHandle	564
	notebook	563	~IPageSelectEvent	563
	pageHandle	563		

The IPageSelectEvent class represents a notification that a page has been selected in a notebook. A notebook page handler creates and uses an object of this class.

Note: This event is first dispatched to the notebook and then to the owner window of the notebook.

Public Functions

Constructors

Although you can create objects of this class, typically IPageHandler::dispatchHandlerEvent (p. 553) creates objects of this class from an object of the class IEvent (Vol. II), IControlEvent (Vol. II), or IPageEvent (p. 546).

IPageSelectEvent

	IPageSelectEvent(const IEvent& event);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		Y	Y	Y

event Reference to an event object.

Use this version of the constructor to create a page select event object from an event object.

IPageSelectEvent

2	IPageSelectEvent(const IControlEvent& controlEvent);	<table border="0"> <tr> <td><u>Win</u></td> <td><u>PM</u></td> <td><u>Motif</u></td> </tr> <tr> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	Y						

controlEvent

Reference to a control event object.

Use this version of the constructor to create a page select event object from a control event object.

3	<code>IPageSelectEvent(const IPageEvent& pageEvent);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
		<i>Y</i>	<i>Y</i>	<i>Y</i>

pageEvent Reference to a page event object.

Use this version of the constructor to create a page select event object from a page event object.

~IPageSelectEvent

<code>virtual</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>~IPageSelectEvent();</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

Page Information

Use these functions to obtain information about the selected page, previously selected page, and the notebook.

notebook Returns a pointer to the notebook object that the page belongs to.

virtual INotebook*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
notebook() const;	Y	Y	Y

pageHandle	Returns the handle of the selected notebook page for overrides of <code>IPageHandler::select</code> (p. 554) and <code>IPageHandler::selectPending</code> (p. 554).
-------------------	---

virtual IPageHandle	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pageHandle() const;	Y	Y	Y

Win The Windows tab control implementation always returns the page handle for the previously selected page for an override of `IPageHandler::selectPending` (p. 554). The tab control does not communicate the selected page in the selection pending notification.

IPageSelectEvent

previousSelectedPageHandle

Returns the handle of the previously selected notebook page for overrides of
IPageHandler::select (p. 554) and IPageHandler::selectPending (p. 554).

IPageHandle		<u>Win</u>	<u>PM</u>	<u>Motif</u>
previousSelectedPageHandle() const;		Y	Y	Y

Inherited Public Functions

IPageEvent		
notebook	pageHandle	

IControlEvent		
controlId		

IEvent		
controlHandle	operator =	setDispatchingHandle
controlWindow	parameter1	setEventType
dispatchingWindow	parameter2	setHandle
eventId	passToOwner	setPassToOwner
eventType	result	setResult
handle	setControlHandle	window

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ISetCanvas

Derivation

- IBase
- IVBase
- INotifier
- IWindow
- IControl
- ICanvas
- ISetCanvas

Inherited By

- IToolBar
- IToolBarContainer

Header File isetcv.hpp

Members	Member	Page	Member	Page
	Constructor	566	packTight	577
	alignment	569	packType	569
	border	575	pad	570
	bottomAlign	575	position	572
	centerAlign	575	rightAlign	577
	centerVerticalAlign	576	setAlignment	570
	classDefaultStyle	576	setDeckCount	568
	convertToGUIStyle	571	setDeckOrientation	568
	deckCount	568	setDefaultStyle	571
	deckOrientation	568	setGroupPad	570
	deckOrientationId	575	setLayoutDistorted	569
	decksByGroup	576	setMargin	570
	defaultStyle	571	setPackType	570
	expandForText	574	setPad	571
	groupPad	569	setText	572
	horizontalDecks	576	size	573
	layout	574	text	572
	leftAlign	577	textId	575
	margin	569	topAlign	578
	moveSizeTo	572	topHandle	568
	packEven	577	verticalDecks	578
	packExpanded	577	~ISetCanvas	567

The ISetCanvas class provides 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

ISetCanvas

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 (Vol. II) and IWindow::calcMinimumSize (Vol. II) to size the windows.

This class provides you with a number of alignment and packing options for positioning differently sized child windows.

Note: To have a group box appear around the contents of a set canvas, use the function ISetCanvas::setText (p. 572), in place of creating an IGroupBox (Vol. II) object.

Public Functions

Constructors

You can construct and destruct objects of this class. You cannot copy or assign ISetCanvas objects because both the copy constructor and assignment operator are private functions.

ISetCanvas

```
ISetCanvas( unsigned long windowIdentifier,           Win PM Motif
            IWindow* parent,                         Y   Y   Y
            IWindow* owner,
            const IRectangle& initial = IRectangle ( ),
            const Style& style = defaultStyle ( ) );
```

windowIdentifier

The window identifier of the canvas you are constructing.

We recommend that you do the following:

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

ISetCanvas

Presentation Manager Notes: Do not use FID_XXX values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

parent The parent window of the canvas you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.

owner The owner window of the canvas you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.

Motif Notes: The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.

initial The initial position and size of the canvas you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.

style The window's characteristics. This value can be a combination of ISetCanvas::Style (p. 574), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects. Optional.

Exceptions	
InvalidParameter	<i>style</i> contains an invalid combination of styles. <i>style</i> cannot contain more than one deck orientation style (ISetCanvas::horizontalDecks and ISetCanvas::verticalDecks), more than one pack style (ISetCanvas::packTight, ISetCanvas::packEven, and ISetCanvas::packExpanded), more than one horizontal alignment style (ISetCanvas::leftAlign, ISetCanvas::centerAlign, and ISetCanvas::rightAlign), or more than one vertical alignment style (ISetCanvas::topAlign, ISetCanvas::centerVerticalAlign, and ISetCanvas::bottomAlign).

~ISetCanvas

```
virtual  
~ISetCanvas();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Decks

Use these members to set and query whether child windows should be laid out horizontally in rows or vertically in columns.

ISetCanvas

deckCount 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.

```
unsigned long
    deckCount() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

deckOrientation Returns an enumerator for deck direction. The returned value is an enumerator provided by DeckOrientation (p. 580).

```
DeckOrientation
    deckOrientation() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

setDeckCount Sets the maximum number of decks used by the canvas. This function has no effect if the ISetCanvas::decksByGroup (p. 576) style is being used.

```
virtual ISetCanvas&
    setDeckCount( unsigned long decks );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

setDeckOrientation Sets the direction of the decks.

```
virtual ISetCanvas&
    setDeckOrientation( DeckOrientation value );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

value Use the enumeration DeckOrientation (p. 580) to specify the direction.

Exceptions	
InvalidParameter	You specified an uninitialized ISetCanvas::DeckOrientation value.

Event Handling Implementation

Event-handling implementation members perform processing needed to allow event handlers to properly receive and route events.

topHandle Returns the top window system control for this compound control.

ISetCanvas

```
virtual IWindowHandle  
topHandle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>N</i>	<i>Y</i>

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

setLayoutDistorted

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

```
virtual ISetCanvas&  
setLayoutDistorted( unsigned long layoutAttributesOn,  
                    unsigned long layoutAttributesOff );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Spacing of Child Windows

Use these members to specify the spacing added around and between child windows.

alignment Returns the enumerator for deck alignment. The returned value is an enumerator provided by Alignment (p. 579).

```
Alignment  
alignment() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

groupPad Returns the pad width or height, which is the space before a child window in a deck with a style of IControl::group.

```
unsigned long  
groupPad() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

margin Returns the margin width and height, which is the space between the edge of the canvas and the outermost child windows.

```
ISize  
margin() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

packType Returns an enumerator for child window spacing in decks. The returned value is an enumerator provided by PackType (p. 580).

ISetCanvas

PackType	<u>Win</u>	<u>PM</u>	<u>Motif</u>
packType() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

pad Returns the pad width and height, which is the space between child windows in a deck and between multiple decks.

ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
pad() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

setAlignment Sets how child windows are aligned in decks.

Note: If the canvas uses a pack type of expanded, this function has no effect.

virtual ISetCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setAlignment(Alignment value);	<i>Y</i>	<i>Y</i>	<i>Y</i>

value Use the enumeration Alignment (p. 579) to specify the child window alignment.

Exceptions	
InvalidParameter	You specified an uninitialized ISetCanvas::Alignment value.

setGroupPad Sets the pad width or height to use for controls with a style of IControl::group. The default value is zero.

virtual ISetCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setGroupPad(unsigned long groupPad);	<i>Y</i>	<i>Y</i>	<i>Y</i>

setMargin Sets the margin width and height, which is the space between the edge of the canvas and the outermost child windows.

virtual ISetCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setMargin(const ISize& margin);	<i>Y</i>	<i>Y</i>	<i>Y</i>

setPackType Sets how child windows are spaced in decks.

virtual ISetCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setPackType(PackType value);	<i>Y</i>	<i>Y</i>	<i>Y</i>

value Use the enumeration PackType (p. 580) to specify the spacing.

ISetCanvas

Exceptions	
InvalidParameter	You specified an uninitialized ISetCanvas::PackType value.

setPad Sets the pad width and height, which is the space between child windows in a deck and between multiple decks.

<pre>virtual ISetCanvas& setPad(const ISize& pad);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an ISetCanvas object, you can use ISetCanvas, ICanvas, and IWindow member functions to query and change individual styles.

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, will be returned if you set *extendedOnly* to true.

<pre>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	N					

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 576) unless you have changed the style using setDefaultStyle (p. 571).

<pre>static Style defaultStyle();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

setDefaultStyle

Sets the default style for all subsequent set canvases.

<pre>static void setDefaultStyle(const Style& style);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>Y</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	Y
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
Y	Y	Y					

ISetCanvas

style A combination of ISetCanvas::Style (p. 574), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects.

Text

Use these members to add a group box around the set canvas or remove it, and to query the text of the group box.

setText 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.

1	<code>virtual ISetCanvas& setText(const char* text);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

text The new group-box text. If you specify a null pointer or a zero-length string, an existing group box is removed.

Motif This function has no effect if the set canvas was constructed without the border style.

2	<code>virtual ISetCanvas& setText(const IResourceId& text);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>Y</i>						

text The identifier of the string table resource for new group-box text. If you specify a zero-length string, an existing group box is removed.

Motif This function has no effect if the set canvas was constructed without the border style.

text Returns the text of the group box. If the canvas has no surrounding group box, an empty IString (Vol. I) is returned.

<code>virtual IString text() const;</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Window Positioning

Use these members to set and query the size and position of windows.

moveSizeTo Changes the position and size of the window.

<code>virtual ISetCanvas& moveSizeTo(const IRectangle& aRectangle);</code>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>N</i></td><td><i>N</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>N</i>	<i>N</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>N</i>	<i>N</i>	<i>Y</i>					

position Returns the position of the window.

ISetCanvas

```
virtual IPoint  
    position() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>N</i>	<i>Y</i>

size Returns the size of the window.

```
virtual ISize  
    size() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>N</i>	<i>N</i>	<i>Y</i>

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

ISetCanvas

layout Positions and sizes child windows.

<pre>virtual ISetCanvas& layout();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Text

Use these members to add a group box around the set canvas or remove it, and to query the text of the group box.

expandForText

Determines how much larger the canvas must be to include a group box with text.

<pre>virtual ISetCanvas& expandForText();</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>Y</i>

Exceptions	
IAccessError	The call to determine the size required to display the text of the set canvas failed.

Inherited Protected Functions

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

ISetCanvas provides notifications that allow observers to process changes to the window.

ISetCanvas

deckOrientationId

Notification identifier provided to observers when the deck orientation of the set canvas changes. ISetCanvas provides the new deck orientation value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). This value is one of the predefined DeckOrientation enumeration values.

static INotificationId const deckOrientationId;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

textId

Notification identifier provided to observers when the text of the set canvas changes.

static INotificationId const textId;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an ISetCanvas object, you can use ISetCanvas, ICanvas, and IWindow member functions to query and change individual styles.

border

Draws a border around the set canvas. If you call the function setText (p. 572), the text you specify will be placed in the border.

static const Style border;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	N	N	Y

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.

static const Style bottomAlign;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

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.

ISetCanvas

If you use this style with `packExpanded`, it is ignored. You cannot specify this style with `leftAlign` or `rightAlign`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>centerAlign;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

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`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>centerVerticalAlign;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

classDefaultStyle

Specifies the original default style for this class, which is `ISetCanvas::horizontalDecks` | `ISetCanvas::packTight` | `ISetCanvas::leftAlign` | `ISetCanvas::topAlign` | `IWindow::visible`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>classDefaultStyle;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

decksByGroup

Causes the number of decks in the set canvas to be determined by the number of child windows with the group style (`IWindow::isGroup` returns true). Each child window with the group style starts a new deck. If you specify this style, calls to `setDeckCount` are ignored.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>decksByGroup;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

horizontalDecks

Organizes child windows in rows. Multiple rows are arranged from top to bottom.

Note: You cannot specify this style with `ISetCanvas::verticalDecks` (p. 578).

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>horizontalDecks;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

ISetCanvas

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`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>leftAlign;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

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`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>packEven;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

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`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>packExpanded;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

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`.

<code>static const Style</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
<code>packTight;</code>	<i>Y</i>	<i>Y</i>	<i>Y</i>

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`.

ISetCanvas

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
rightAlign;	Y	Y	Y

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.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
topAlign;	Y	Y	Y

verticalDecks Organizes child windows in columns. Multiple columns are arranged from left to right.

Note: You cannot specify this style with ISetCanvas::horizontalDecks (p. 576).

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
verticalDecks;	Y	Y	Y

Inherited Public Data

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ISetCanvas contains the following nested classes:

ISetCanvas::Style (see page 581)

Alignment

```
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.

ISetCanvas

bottomRight

Aligns the child windows in the canvas to the bottom and right edge in the deck.

DeckOrientation

```
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

```
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 effect 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.



ISetCanvas::Style

Derivation IBase
 IBitFlag
 ISetCanvas::Style

Inherited By None.

Header File isetcv.hpp

The nested class ISetCanvas::Style provides a set of valid canvas styles for the ISetCanvas::defaultStyle (p. 571) and ISetCanvas::setDefaultStyle (p. 571) functions, and for the constructor of the ISetCanvas (p. 565) class. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
 IWindow::Style (Vol. II)

Once you have constructed an ISetCanvas object, you can use ISetCanvas, ICanvas, and IWindow member functions to query and change individual styles.

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 (p. 570), setDeckOrientation (p. 568), and setPackType (p. 570) member functions of the ISetCanvas class.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

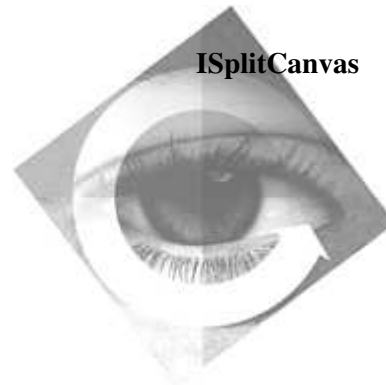
Inherited Protected Functions

ISetCanvas::Style

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



ISplitCanvas

Derivation

```

IBase
IVBase
INotifier
IWindow
IControl
ICanvas
ISplitCanvas
  
```

Inherited By None.

Header File isplitcv.hpp

Members	Member	Page	Member	Page
	Constructor	587	setOrientation	589
	classDefaultStyle	593	setSplitBarEdgeColor	586
	convertToGUIStyle	590	setSplitBarMiddleColor	586
	defaultStyle	590	setSplitBarThickness	589
	horizontal	593	setSplitWindowPercentage	585
	layout	591	splitBarEdgeColor	587
	noSplitBars	593	splitBarMiddleColor	587
	orientation	589	splitBarThickness	590
	orientationId	592	splitWindowPercentage	586
	resetSplitBarEdgeColor	586	vertical	593
	resetSplitBarMiddleColor	586	visibleRectangle	589
	setDefaultStyle	590	~ISplitCanvas	588
	setLayoutDistorted	588		

The ISplitCanvas class provides 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 (Vol. II) 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 for its parent.

ISplitCanvas

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 multicell 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 `IControl::tabStop` (Vol. II).

To specify what percentage of the split canvas a control occupies when the split canvas is initially displayed, use `ISplitCanvas::setSplitWindowPercentage` (p. 585). 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.

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 `ISplitCanvas::noSplitBars` (p. 593). 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.

ISplitCanvas

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 does not support the enumeration `SplitBarEdge`. The value of the edge is always zero. The enumerator `splitBarMiddle` specifies the thickness of the split bar.

Public Functions

Child Window Sizing

Use these members to control the size of the child windows of the split canvas through percentages (ratios).

setSplitWindowPercentage

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 or 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` (Vol. II) to update the appearance of the canvas.

```
virtual ISplitCanvas&
    setSplitWindowPercentage( IWindow* window,
                             unsigned long percentage );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

ISplitCanvas

window The child window you are indirectly sizing.
percentage The portion of the split window that the child window will occupy.

splitWindowPercentage

Returns the percentage of the width or height of the canvas currently occupied by the specified window.

unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
splitWindowPercentage(IWindow* window);	<i>Y</i>	<i>Y</i>	<i>Y</i>
<i>window</i>	The child window.		

Colors

Use these members to set and query the color of the middle or edge of a split bar.

resetSplitBarEdgeColor

Resets the color of the edges of the canvas' split bars, so the default edge color is used.

virtual ISplitCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetSplitBarEdgeColor();	<i>Y</i>	<i>Y</i>	<i>I</i>

resetSplitBarMiddleColor

Resets the color of the interior of the canvas' split bars, so that the default color is used.

virtual ISplitCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
resetSplitBarMiddleColor();	<i>Y</i>	<i>Y</i>	<i>I</i>

setSplitBarEdgeColor

Sets the color for the edges of the canvas' split bars.

virtual ISplitCanvas&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setSplitBarEdgeColor(const IColor& color);	<i>Y</i>	<i>Y</i>	<i>I</i>

setSplitBarMiddleColor

Sets the color for the interior of the canvas' split bars.

ISplitCanvas

```
virtual ISplitCanvas&
    setSplitBarMiddleColor( const IColor& color );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

splitBarEdgeColor

Returns the color of the edges of the canvas' split bars.

```
virtual IColor
    splitBarEdgeColor() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>I</i>

splitBarMiddleColor

Returns the color of the interior of the canvas' split bars.

```
virtual IColor
    splitBarMiddleColor() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Constructors

You can construct and destruct objects of this class. You cannot copy or assign ISplitCanvas objects because both the copy constructor and assignment operator are private functions.

ISplitCanvas

```
ISplitCanvas( unsigned long windowIdentifier,
               IWindow* parent,
               IWindow* owner,
               const IRectangle& initialSize = IRectangle ( ),
               const Style& style = defaultStyle ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

windowIdentifier

The window identifier of the canvas you are constructing.

We recommend that you do the following:

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

Presentation Manager Notes: Do not use FID_xxx values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

ISplitCanvas

- parent

The parent window of the canvas you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.
- owner

The owner window of the canvas you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.
Motif Notes: The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.
- initialSize

The initial position and size of the canvas you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.
- style

The window’s characteristics. This value can be a combination of ISplitCanvas::Style (p. 592), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects. Optional.

Exceptions	
InvalidParameter	style contains an invalid combination of styles. style cannot contain both the ISplitCanvas::horizontal (p. 593) and ISplitCanvas::vertical (p. 593) styles.

ISplitCanvas

virtual			
ISplitCanvas();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	Y

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

setLayoutDistorted

Treats the following as changes to the layout of its child windows:

- Changing the size of the canvas
- Adding a child window
- Removing a child window

virtual ISplitCanvas&			
setLayoutDistorted(unsigned long layoutAttributesOn,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
unsigned long layoutAttributesOff);	Y	Y	Y

Orientation

These members control the orientation of the canvas' split bars.

orientation Returns an enumerator Orientation (p. 594) for the orientation of the canvas' split bars.

Orientation orientation() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y Y
-------------------------------------	--

setOrientation

Sets the direction in which the canvas' split bars are drawn.

virtual ISplitCanvas& setOrientation(Orientation value);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y Y
---	--

value Use the enumeration Orientation (p. 594) to specify the direction.

Painting Optimization

These members provide a parent window with information that allows it to refresh the screen more efficiently.

visibleRectangle

Returns the portion of the window that the canvas actually paints. The parent window can perform painting optimizations by not painting this area.

virtual IRectangle visibleRectangle() const;	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y N
---	--

Split Bar Thickness

Use these members to customize the thickness of the split bar.

setSplitBarThickness

Sets the thickness of the specified area of the split bar.

virtual ISplitCanvas& setSplitBarThickness(SplitBarArea area, unsigned long thickness);	<u>Win</u> <u>PM</u> <u>Motif</u> Y Y Y
---	--

value Use the enumeration SplitBarArea (p. 594) to specify the area.

ISplitCanvas

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.

splitBarThickness

Returns the thickness of the specified area of the split bar.

unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
splitBarThickness(SplitBarArea area);		<i>Y</i>	<i>Y</i>	<i>Y</i>
<i>value</i>	Use the enumeration SplitBarArea (p. 594) to specify the area of the split bar.			

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an ISplitCanvas object, you can use ISplitCanvas, ICanvas, and IWindow member functions to query and change individual styles.

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, will be returned if you set *extendedOnly* to true.

virtual unsigned long		<u>Win</u>	<u>PM</u>	<u>Motif</u>
convertToGUIStyle(const IBitFlag& style,		<i>Y</i>	<i>Y</i>	<i>N</i>
Boolean extendedOnly = false) const;				

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 593) unless you have changed the style using setDefaultStyle (p. 590).

static Style		<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultStyle();		<i>Y</i>	<i>Y</i>	<i>Y</i>

setDefaultStyle

Sets the default style for all subsequent split canvases.

ISplitCanvas

```
static void  
    setDefaultStyle( const Style& style );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

style A combination of ISplitCanvas::Style (p. 592), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects.

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

layout Computes the position and size of all child controls.

ISplitCanvas

virtual ISplitCanvas&
layout();

Win

PM

Motif

Y

Y

Y

Inherited Protected Functions

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

These members allow observers to process changes to the window.

orientationId Notification identifier provided to observers when the orientation of the split canvas changes. ISplitCanvas provides the new split canvas orientation value in the INotificationEvent::eventData (Vol. I) field of the INotificationEvent (Vol. I). as an Orientation enumeration.

static INotificationId const
orientationId;

Win

PM

Motif

Y

Y

Y

Styles

Use these style members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

- ICanvas::Style (p. 35)
- IWindow::Style (Vol. II)

Once you have constructed an ISplitCanvas object, you can use ISplitCanvas, ICanvas, and IWindow member functions to query and change individual styles.

ISplitCanvas

classDefaultStyle

Specifies the original default style for this class, which is ISplitCanvas::vertical | IWindow::visible.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

horizontal

Draws split bars horizontally, placing the panes top to bottom.

Note: You cannot specify this style with ISplitCanvas::vertical (p. 593).

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
horizontal;	Y	Y	Y

noSplitBars

Specifies no split bars are used to separate panes.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noSplitBars;	Y	Y	Y

vertical

Draws split bars vertically, placing the panes left to right. This style is the default.

Note: You cannot specify this style with ISplitCanvas::horizontal (p. 593).

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
vertical;	Y	Y	Y

Inherited Public Data

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId

ISplitCanvas

IWindow		
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

ISplitCanvas contains the following nested classes:

ISplitCanvas::Style (see page 595)

Orientation Orientation {
 horizontalSplit,
 verticalSplit
 };

Use these enumerators to specify the direction of the canvas's split bars:

horizontalSplit

Draws split bars horizontally, placing the panes top to bottom.

verticalSplit

Draws split bars vertically, placing the panes left to right.

SplitBarArea SplitBarArea {
 splitBarEdge,
 splitBarMiddle
 };

Use these enumerators to specify different areas of a 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 interior of a split bar.



ISplitCanvas::Style

Derivation IBase
 IBitFlag
 ISplitCanvas::Style

Inherited By None.

Header File isplitcv.hpp

The nested class ISplitCanvas::Style provides a set of valid split canvas styles for the ISplitCanvas::defaultStyle (p. 590) and ISplitCanvas::setDefaultStyle (p. 590) functions, and for the constructor of the ISplitCanvas (p. 583) class. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
 IWindow::Style (Vol. II)

Once you have constructed an ISplitCanvas object, you can use ISplitCanvas, ICanvas, and IWindow member functions to query and change individual styles.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

ISplitCanvas::Style

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IStringGenerator

Derivation IBase
IStringGenerator

Inherited By None.

Header File istrngen.hpp

Members	Member	Page	Member	Page
	Constructor	598	stringFor	599
	operator =	598	~IStringGenerator	599

The IStringGenerator class manages the translation of objects to their IString form. Use the IStringGenerator::stringFor member function to translate an object to its IString representation. Call it on an object-by-object basis. In the default case (no IStringGeneratorFn (p. 601)), the object should support the operator-> or be an Object*, where the Object class has an asString member function.

The IStringGenerator::stringFor member function processes as follows:

1. If no IStringGeneratorFn object has been set (the default), then it uses an object's asString member function for translation.
2. If an optional IStringGeneratorFn has been set, it uses the IStringGeneratorFn::stringFor member function.

Note: IStringGeneratorFn objects represent functions. Objects of the IStringGeneratorMemberFn (p. 604) and IStringGeneratorRefMemberFn (p. 607) classes represent member functions and differ in how they treat the object parameter in the stringFor member function.

Create and use IStringGenerator objects by following these steps:

- Create an IStringGeneratorFn derived object containing the function to be called to generate an IString.
- Create an IStringGenerator object initializing it with the IStringGeneratorFn derived object.
- Call the IStringGenerator::stringFor member function passing as the parameter the object you want a string generated for.

IStringGenerator

Note: IStringGeneratorFn, as an abstract base class, establishes the protocol for specifying a function to translate an object to an IString. Inherit from this class and implement the stringFor member function to define the calling convention and call the stored function.

Public Functions

Constructors

You can construct and destruct objects of this class. The copy constructor and operator= ensure that the contained generator function objects are reference-counted properly.

IStringGenerator

1	IStringGenerator(const IReference < IStringGeneratorFn < Element > >& generatorFunction = 0);	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

You create a string generator object using an optional IStringGeneratorFn.

generatorFunction

A pointer (IReference) to an existing IStringGeneratorFn-derived object. If it is specified, subsequent calls to stringFor will use this function to produce strings for the given object.

Note: The pointer, IReference, object ensures the generator function (which is of type IRefCounted) is reference-counted properly.

2	IStringGenerator(const IStringGenerator < Element >& stringGenerator);	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td>Y</td><td>Y</td><td>N</td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	Y	Y	N
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
Y	Y	N						

You can create a string generator object using another IStringGenerator. This ensures that the reference-counting of the IStringGeneratorFn object is done properly.

stringGenerator

A string generator.

operator = Assign one string generator to another. This ensures that the reference-counting of the IStringGeneratorFn object is done properly.

stringGenerator

A string generator.

IStringGenerator

```
IStringGenerator < Element >&
operator =(
    const IStringGenerator < Element >& stringGenerator);
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IStringGenerator

```
virtual
~IStringGenerator();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Generate String

Use this member to translate an object to its IString form.

stringFor Obtains an IString version of the element. If a local IStringGeneratorFn has been set, then it is used. Otherwise, the element's asString member function generates the IString.

element Reference to the object for which it generates the string.

Note: For the default case (no IStringGeneratorFn), the element should natively support operator-> or the element must be an Object*, where the Object class has an asString member function.

IStringGeneratorRefMemberFn and IStringGeneratorMemberFn (IStringGeneratorFn derived classes) contain member functions that take no parameters. They support invoking these member functions on Object* elements (element is a pointer) and concrete Object elements (element is not a pointer), respectively. If you need additional string generation support, create your own IStringGeneratorFn derived class.

```
IString
stringFor( const Element& element ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

IStringGenerator

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IStringGeneratorFn

Derivation	IBase
	IVBase
	IRefCounted
	IStringGeneratorFn
Inherited By	IStringGeneratorMemberFn
	IStringGeneratorRefMemberFn

Header File istrngen.hpp

Members	Member	Page
	Constructor	602
	stringFor	602
	~IStringGeneratorFn	602

The IStringGeneratorFn template class is an abstract base class defining the protocol for storing and calling functions to generate IStrings. Objects of this class represent functions to be called when the stringFor function is called. The stringFor pure virtual function accepts an object reference of the template class type.

Note: IStringGenerator (p. 597) does the following:

- Uses these objects (calls IStringGeneratorFn::stringFor) to generate IStrings
- Accepts an optional parameter, a reference to an IStringGeneratorFn on its constructor

Use the subclasses IStringGeneratorMemberFn and IStringGeneratorRefMemberFn or create your own subclass to represent functions. These subclasses do two things:

- On the constructor, save a pointer to the function the object represents
- Implement stringFor to call the stored function

These objects are reference-counted to manage their destruction.

IStringGeneratorMemberFn and IStringGeneratorRefMemberFn objects represent member functions of the template class type. Review these classes for example usage.

IStringGeneratorFn

Public Functions

Constructors

You do not create objects of this class. It is an abstract base class.

IStringGeneratorFn

```
IStringGeneratorFn();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

~IStringGeneratorFn

```
virtual  
~IStringGeneratorFn();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Generate String

Establishes the protocol for generating an IString for the object parameter. Subclasses implement this function to generate an IString for that object.

stringFor

Is a pure virtual function. It accepts an object reference as a parameter; it is this object for which subclass objects generate a string. IStringGeneratorFn subclasses override this function to call the appropriate member function with the appropriate number of parameters.

```
virtual IString  
stringFor( const T& object ) = 0;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile

IStringGeneratorFn

IBase		
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	

IStringGeneratorMemberFn

IStringGeneratorMemberFn

Derivation

```
graph TD
    IBase --> IVBase
    IVBase --> IRefCounted
    IRefCounted --> IStringGeneratorFn
    IStringGeneratorFn --> IStringGeneratorMemberFn
```

Inherited By None.

Header File istrngen.hpp

Members

Member	Page
Constructor	605
stringFor	605
~IStringGeneratorMemberFn	605

The IStringGeneratorMemberFn template class is an IStringGeneratorFn-derived class. It dispatches C++ member functions against an object to generate an IString. These objects represent member functions of class **T**, where **T** is the template argument. Objects of this class apply the stored member function against class **T** objects in the stringFor member function, where object is a reference to a class **T** object.

The constructor for these objects requires a pointer to a class **T** member function; this member function returns an IString and accepts no arguments. Subclass this class to support member functions with additional parameters.

The following example creates an IStringGeneratorMemberFn object:

```
class MyClass {
public:
    IString myString()
    {
        // Code to generate a string
    }
    // .....
};
//...
MyClass myObj;
IStringGeneratorMemberFn<MyClass> * genFunction = new
    IStringGeneratorMemberFn<MyClass>( MyClass::myString );
```

Note: Objects of the IStringGeneratorMemeberFn class require the object passed on the stringFor member function not be a pointer.

IStringGeneratorMemberFn

IStringGeneratorRefMemberFn objects generate strings from pointers to objects.

Public Functions

Constructors

You can construct and destruct objects of this class.

IStringGeneratorMemberFn

Create from a pointer to a member function of class **T**, where **T** is the template argument. The member function must return an IString and accept no arguments; also, it is not a const function.

member A pointer to the member function.

IStringGeneratorMemberFn(IString (T::* member) ());	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

~IStringGeneratorMemberFn

virtual ~IStringGeneratorMemberFn();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Generate String

Use this member to generate an IString from an object.

stringFor

Calls the appropriate member function on the object reference parameter. The called member function is initialized from the parameter on the IStringGeneratorMemberFn constructor.

object Reference to the object against which it dispatches the member function.

Note: The object parameter must be an object of type **T** (where **T** is the template class argument) and not a pointer.

virtual IString stringFor(const T& object);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IStringGeneratorMemberFn

Inherited Public Functions

IStringGeneratorFn		
stringFor		

IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IStringGeneratorRefMemberFn

Derivation IBase
 IVBase
 IRefCounted
 IStringGeneratorFn
 IStringGeneratorRefMemberFn

Inherited By None.

Header File istrngen.hpp

Members	Member	Page
	Constructor	608
	stringFor	608
	~IStringGeneratorRefMemberFn	608

The IStringGeneratorRefMemberFn template class is an IStringGeneratorFn-derived class. It dispatches C++ member functions against an object pointer to generate an IString. These objects represent member functions of class U to be applied against objects of type T, where T and U are template arguments and T is defined as typedef U* T;. This makes objects of type T the pointers to objects of class U. Objects of this class apply the stored member function against type T objects in the stringFor(const T& object) function, where object is a reference to a type T object.

The constructor for these objects requires a pointer to a class U member function; this member function returns an IString and accepts no arguments. Derive a class from this class to support member functions with additional parameters.

The following example creates an IStringGeneratorRefMemberFn object:

```
class MyClass {
public:
    IString myString()
    {
        // Code to generate a string
    }
    // .....
};
//...
typedef MyClass * pMyClass;
MyClass myObj;
IStringGeneratorRefMemberFn<MyClass> * genFunction = new
    IStringGeneratorRefMemberFn<MyClass>( pMyClass, MyClass::myString );
```

IStringGeneratorRefMemberFn

Note: Objects of the IStringGeneratorRefMemberFn class require that the object passed on the stringFor member function is a pointer or an object that supports the operator->. IStringGeneratorMemberFn objects generate strings from objects that are not pointers.

Public Functions

Constructors

You can construct and destruct objects of this class.

IStringGeneratorRefMemberFn

Create from a pointer to a member function of class **U**, where **U** is a template argument. The member function must return an IString, accept no arguments, and be a non-const function.

member A pointer to the member function.

```
IStringGeneratorRefMemberFn( IString ( U::* member ) ( ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

This constructor requires a pointer to a class **U** member function. This member function should return an IString, accept no parameters and be a non-const function.

~IStringGeneratorRefMemberFn

```
virtual  
~IStringGeneratorRefMemberFn();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Generate String

Use this member to generate an IString from an object pointer.

stringFor

This function calls the appropriate member function on the object reference parameter. The called member function is initialized from the parameter on the IStringGeneratorMemberFn constructor. The object parameter should be a pointer.

object Reference to the object against which it dispatches the member function.

Note: The object parameter must be an object of type **T**, which is a pointer to an object of type **U** (where **T** and **U** are the template class arguments) and class **U** contains the member function that is dispatched.

IStringGeneratorRefMemberFn

```
virtual IString  
    stringFor( const T& object );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IStringGeneratorFn		
stringFor		

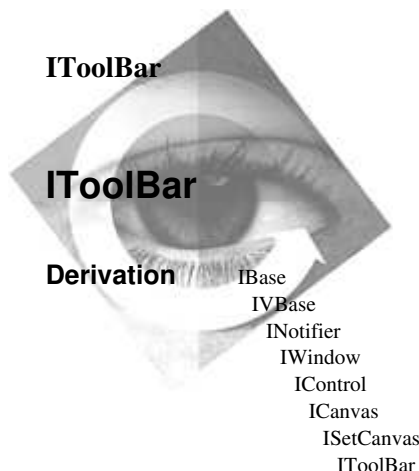
IRefCounted		
addRef	removeRef	useCount

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File itbar.hpp

Members				
	Member	Page	Member	Page
	Constructor	611	floatingFrame	614
	addAsFirst	617	floatingPosition	614
	addAsLast	617	floatingTitle	614
	addAsNext	617	initialize	622
	addAsPrevious	617	isExpanded	613
	allowsDragDrop	612	isMisfitFilteringEnabled	614
	buttonBitmapAndTextVisible	623	location	620
	buttonBitmapVisible	623	moveAfter	617
	buttonTextVisible	624	moveBefore	618
	buttonView	616	moveToFirst	618
	calcMinimumSize	622	moveToLast	618
	classDefaultStyle	624	noDragDrop	624
	collapse	613	remove	618
	convertToGUIStyle	615	setButtonView	616
	createFloatingFrame	622	setDefaultGroupPad	619
	createToolBarContainer	622	setDefaultMargin	619
	defaultGroupPad	619	setDefaultMisfitWidth	620
	defaultMargin	619	setDefaultPad	620
	defaultMisfitWidth	619	setDefaultStyle	616
	defaultPad	619	setFloatingPosition	614
	defaultStyle	615	setFloatingTitle	615
	disableDragDrop	612	setLayoutDistorted	611
	disableMisfitFiltering	613	setLocation	620
	enableDragDrop	612	toolBarAt	612
	enableMisfitFiltering	613	toolBarContainer	616
	expand	613	windowAt	612
	filter	622	~IToolBar	611
	filterMisfits	624		

The IToolBar class creates and manages a tool bar for a frame window.

Public Functions

Canvas Layout

Use these members to provide support for canvas layout.

setLayoutDistorted

Treats a minimum size change for a child window like a layout change. This function optimizes the layout of the tool bar during initialization.

virtual IToolBar&		<u>Win</u>	<u>PM</u>	<u>Motif</u>
setLayoutDistorted(unsigned long layoutAttributeOn,		<i>Y</i>	<i>Y</i>	<i>N</i>
unsigned long layoutAttributeOff);				

Constructors

You can construct and destruct objects of this class. You cannot copy or assign `IToolBar` objects because both the copy constructor and assignment operator are private functions.

IToolBar

I	IToolBar(unsigned long identifier, IFrameWindow* owner, Location location = aboveClient, Boolean groupWithPreceding = false, const Style& style = defaultStyle ());	Win <i>Y</i>	PM <i>Y</i>	Motif <i>N</i>
----------	--	------------------------	-----------------------	--------------------------

Creates a tool bar as the last tool bar in the tool bar area defined by the frameLocation.

If *groupWithPreceding* is true, the tool bar is placed on the same row or column as the last tool bar in the specified frame location (if one exists). If an `IToolBarController` is needed for the location indicated, it is created automatically when you use this constructor.

2	<pre> IToolBar(unsigned long identifier, IToolBar* precedingToolBar, Boolean groupWithPreceding = false, const Style& style = defaultStyle ()); </pre>	<table border="0"> <tr> <td style="text-align: right;"><u>Win</u></td> <td style="text-align: right;"><u>PM</u></td> <td style="text-align: right;"><u>Motif</u></td> </tr> <tr> <td style="text-align: right;"><i>Y</i></td> <td style="text-align: right;"><i>Y</i></td> <td style="text-align: right;"><i>N</i></td> </tr> </table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>						
<i>Y</i>	<i>Y</i>	<i>N</i>						

Creates a tool bar relative to an existing tool bar.

~IToolBar

IToolBar

```
virtual  
~IToolBar();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Cursor Members

Use these members to query the IToolBar at the current FrameCursor position or the IWindow at the current WindowCursor position.

toolBarAt Returns a pointer to the IToolBar at the cursor position.

```
static IToolBar*  
toolBarAt( const FrameCursor& cursor );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Exceptions	
InvalidParameter	An invalid cursor was specified.

windowAt Returns a pointer to the IWindow at the cursor position.

```
IWindow*  
windowAt( const WindowCursor& cursor ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Drag and Drop Support

Use these members to set and query whether the tool bar supports drag and drop.

allowsDragDrop

Returns true if the tool bar supports drag and drop.

```
Boolean  
allowsDragDrop() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

disableDragDrop

Prevents dragging of the tool bar, as well as the objects that reside on the tool bar, such as a tool bar button. Prevents drops from occurring on the tool bar.

```
virtual IToolBar&  
disableDragDrop();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

enableDragDrop

Enables dragging of the tool bar, as well as the objects that reside on the tool bar, such as a tool bar button. Enables drops on the tool bar.

IToolBar

```
virtual IToolBar&  
    enableDragDrop( Boolean enable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Expanding and Collapsing the Tool Bar

Use these members to expand or collapse the height of the tool bar when it is floating. If the tool bar is collapsed, only the title bar of the floating frame is displayed.

collapse Displays only the title of tool bar when it is floating.

```
virtual IToolBar&  
    collapse();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

expand Displays the entire tool bar when it is floating.

```
virtual IToolBar&  
    expand( Boolean expand = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

isExpanded Returns true if the entire tool bar is displayed when it is floating.

```
Boolean  
    isExpanded() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Filter Misfits

Use these members to query or change the filterMisfits style for IToolBar objects.

disableMisfitFiltering

Removes the filterMisfits style from the tool bar.

```
IToolBar&  
    disableMisfitFiltering();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

enableMisfitFiltering

Sets the filterMisfits style for the tool bar.

```
IToolBar&  
    enableMisfitFiltering( Boolean enable = true );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBar

isMisfitFilteringEnabled

Returns true if the filterMisfits style is set for the tool bar.

Boolean

isMisfitFilteringEnabled() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Floating Tool Bar

Use these members to set or query information about a floating tool bar.

floatingFrame

Returns the IToolBarFrameWindow object that contains this tool bar. A 0 is returned if this tool bar is not currently in a floating frame window.

virtual IToolBarFrameWindow*
floatingFrame();

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

floatingPosition

Returns the current position of the frame window that encloses the floating tool bar (relative to the owning frame window used to construct the tool bar).

If the tool bar is not floating, the position the tool bar would occupy if it were floating is returned instead.

IPoint
floatingPosition() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

floatingTitle Returns the title text of the floating frame window.

If the tool bar is not floating, the title that would be used if the tool bar were floating is returned instead.

IString
floatingTitle() const;

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setFloatingPosition

Sets the position of the frame window that encloses the floating tool bar. The position is relative to the owning frame window used to construct the tool bar.

If the tool bar is not currently floating, the position is saved and used if the tool bar location is changed to floating.

IToolBar

```
virtual IToolBar&
    setFloatingPosition( const IPoint& frameRelativePosition );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setFloatingTitle

Sets the title text of the floating frame window.

If the tool bar is not floating, the title is saved and used when the tool bar location is changed to floating.

```
1 virtual IToolBar&
    setFloatingTitle( const IResourceId& text );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

```
2 virtual IToolBar&
    setFloatingTitle( const char* text );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Styles

Use these members to set and query tool bar styles. You can use these styles with the styles defined by the following nested classes:

IWindow Styles (Vol. II)
ICanvas Styles (p. 35)
ISetCanvas Styles (p. 574)

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform.

Extended styles, those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

```
virtual unsigned long
    convertToGUIStyle( const IBitFlag& style,
                      Boolean extendedOnly = false ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

defaultStyle Returns the default style. The default style is classDefaultStyle unless you have changed the style using setDefaultStyle.

```
static Style
    defaultStyle();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBar

setDefaultStyle

Sets the default style for all subsequent tool bars.

style Use the styles provided by IToolBar::Style to specify the default style.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultStyle(const Style& style);	<i>Y</i>	<i>Y</i>	<i>N</i>

Tool Bar Button Behavior

Use these members to affect the appearance of any IToolBarButton objects in the tool bar.

buttonView Returns the current view of IToolBarButton objects in the tool bar.

IToolBarButton::View	<u>Win</u>	<u>PM</u>	<u>Motif</u>
buttonView() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setButtonView

Changes the IToolBarButton members of the tool bar to the value of IToolBarButton::view. Valid values are IToolBarButton::bitmapView, IToolBarButton::textView, and IToolBarButton::bitmapAndText.

virtual IToolBar&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setButtonView(IToolBarButton::View buttonView);	<i>Y</i>	<i>Y</i>	<i>N</i>

Tool Bar Container

Use these members to access IToolBarContainer objects.

toolBarContainer

Returns a pointer to the IToolBarContainer object that contains this tool bar. A 0 is returned if this tool bar is not currently in a tool bar container.

virtual IToolBarContainer*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
toolBarContainer() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Tool Bar Contents

Use these members to add and remove items in the tool bar. Explicit addition and removal of IToolBar items is necessary to enable correct direct manipulation behavior and to ensure that misfits are properly filtered in and out of the tool bar.

IToolBar

addAsFirst Adds the window as the first item in the tool bar.

If *startNewGroup* is true, the window is placed by itself into a new group. Otherwise, the window becomes part of the first group in the tool bar.

<pre>virtual IToolBar& addAsFirst(IWindow* window, Boolean startNewGroup = false);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsLast Adds the window as the last item in the tool bar.

If *startNewGroup* is true, the window is placed by itself into a new group. Otherwise, the window becomes part of the last group in the tool bar.

<pre>virtual IToolBar& addAsLast(IWindow* window, Boolean startNewGroup = false);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsNext Adds the window as the next item in the tool bar immediately after the reference window.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the same group as the reference window.

<pre>virtual IToolBar& addAsNext(IWindow* window, IWindow* referenceWindow, Boolean startNewGroup = false);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

addAsPrevious

Adds the window as the previous item in the tool bar immediately before the reference window.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the same group as the reference window.

<pre>virtual IToolBar& addAsPrevious(IWindow* window, IWindow* referenceWindow, Boolean startNewGroup = false);</pre>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

moveAfter Moves the window so that it immediately follows the reference window.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the same group as the reference window.

IToolBar

Note: This function is equivalent to calling the remove and addAsNext methods.

<pre>virtual IToolBar& moveAfter(IWindow* window, IWindow* referenceWindow, Boolean startNewGroup = false);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

moveBefore Moves the window so that it immediately precedes the reference window.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the same group as the reference window.

Note: This function is equivalent to calling the remove and addAsPrevious methods.

<pre>virtual IToolBar& moveBefore(IWindow* window, IWindow* referenceWindow, Boolean startNewGroup = false);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

moveToFirst Moves the window so that it is the first window in the tool bar.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the first group in the tool bar.

Note: This function is equivalent to calling the remove and addAsFirst methods.

<pre>virtual IToolBar& moveToFirst(IWindow* window, Boolean startNewGroup = false);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

moveToLast Moves the window so that it is the last window in the tool bar.

If *startNewGroup* is true, the window starts a new group. Otherwise, the window becomes part of the last group in the tool bar.

Note: This function is equivalent to calling the remove and addAsLast methods.

<pre>virtual IToolBar& moveToLast(IWindow* window, Boolean startNewGroup = false);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

remove Removes the window from the tool bar.

<pre>virtual IToolBar& remove(IWindow* window);</pre>	<table><thead><tr><th><u>Win</u></th><th><u>PM</u></th><th><u>Motif</u></th></tr></thead><tbody><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></tbody></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

IToolBar

Tool Bar Defaults

Use these static members to affect the appearance of all IToolBar objects in the application.

defaultGroupPad

Returns the current number of pels of padding added between groups in tool bars.

static unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultGroupPad();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultMargin

Returns the current number of pels of margin added around windows in tool bars.

static ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultMargin();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultMisfitWidth

Returns the current width used to filter oversized windows from tool bars.

static unsigned long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultMisfitWidth();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultPad

Returns the current default number of pels used to separate windows in tool bars.

static long	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultPad();	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultGroupPad

Sets the number of pels of padding between groups of windows in the tool bar. This pad value is added to any window that returns true from IWindow::isGroup.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultGroupPad(unsigned long pixelsOfGroupPad = 8);	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultMargin

Sets the number of pels of margin around windows in the tool bar. The sizeOfMargin value is applied directly to a horizontal tool bar and rotated for a vertical tool bar. In a horizontal tool bar, sizeOfMargin.width is added to the left and right of the tool bar windows and sizeOfMargin.height is added to the top and bottom of tool bar windows. The initial sizeOfMargin value is ISize(7,4).

IToolBar

```
static void
    setDefaultMargin( const ISize& sizeOfMargin = ISize ( 7 ,      Win PM Motif
                                                                Y   Y   N
                    4 ) );
```

setDefaultMisfitWidth

Sets the width value to be used for filtering windows when a tool bar contains the style IToolBar::filterMisfits.

The initial setDefaultMisfitWidth is 100 pels.

```
static void
    setDefaultMisfitWidth( unsigned long maximumWidth = 100 );  Win PM Motif
                                                                Y   Y   N
```

setDefaultPad

Sets the number of pels of padding between windows added to the tool bar. The initial default is to overlap the members of a group by one pel.

```
static void
    setDefaultPad( long pixelsOfPad = - 1 );  Win PM Motif
                                              Y   Y   N
```

Tool Bar Location

Use these members to set and query the tool bar location.

location Returns the current location of the tool bar.

```
Location
    location() const;  Win PM Motif
                      Y   Y   N
```

setLocation Sets a new location for the tool bar.

```
virtual IToolBar&
    setLocation( Location location );  Win PM Motif
                                      Y   Y   N
```

Inherited Public Functions

ISetCanvas		
alignment	packType	setLayoutDistorted
convertToGUIStyle	pad	setMargin

IToolBar

ISetCanvas		
deckCount	position	setPackType
deckOrientation	setAlignment	setPad
defaultStyle	setDeckCount	setText
groupPad	setDeckOrientation	size
margin	setDefaultStyle	text
moveSizeTo	setGroupPad	topHandle

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Initialization

These members initialize a tool bar.

IToolBar

createFloatingFrame

Creates a new IToolBarFrameWindow to be used for the floating tool bar.

virtual IToolBarFrameWindow*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
createFloatingFrame(IFrameWindow* owner);	<i>Y</i>	<i>Y</i>	<i>N</i>

createToolBarContainer

Creates a new IToolBarContainer to be used for the tool bar.

virtual IToolBarContainer*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
createToolBarContainer(IFrameWindow* frame);	<i>Y</i>	<i>Y</i>	<i>N</i>

filter

Checks if misfits need to be filtered in or filtered out based on the orientation of the tool bar and the current tool bar style.

virtual IToolBar&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
filter();	<i>Y</i>	<i>Y</i>	<i>N</i>

initialize

Initializes the tool bar window when the tool bar needs a tool bar container or floating frame, or when the first child window is added to the toolbar. Initialization is delayed so derived classes can override the createToolBarContainer and createFloatingFrame methods.

virtual IToolBar&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
initialize();	<i>Y</i>	<i>Y</i>	<i>N</i>

Layout

Layout is information used by the canvas classes to provide dialog-like behavior.

calcMinimumSize

Returns the recommended minimum size for this tool bar. If isExpanded is false, the tool bar has a height of 0. Otherwise, this returns the minimum size required to display the contents of the tool bar.

virtual ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
calcMinimumSize() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

ISetCanvas		
expandForText	layout	

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these members to set and query tool bar styles. You can use these styles with the styles defined by the following nested classes:

IWindow Styles (Vol. II)
 ICanvas Styles (p. 35)
 ISetCanvas Styles (p. 574)

buttonBitmapAndTextVisible

Displays both the button text and bitmap for IToolBarButton objects that are added into the tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
buttonBitmapAndTextVisible;	Y	Y	N

buttonBitmapVisible

Displays only the bitmap for IToolBarButton objects that are added into the tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
buttonBitmapVisible;	Y	Y	N

IToolBar

buttonTextVisible

Displays only the button text for IToolBarButton objects that are added into the tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
buttonTextVisible;	<i>Y</i>	<i>Y</i>	<i>N</i>

classDefaultStyle

Specifies the original default style for this class, which is IToolBar::filterMisfits, IToolBar::buttonBitmapVisible, and IWindow::visible.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	<i>Y</i>	<i>Y</i>	<i>N</i>

filterMisfits

Automatically filters out any windows in the tool bar that exceed the misfit width when the tool bar is oriented vertically.

The purpose of this style is to save screen space when a tool bar is oriented vertically. If a wide control is displayed, there would be wasted space to the right of tool bar buttons and other smaller controls.

If a tool bar is oriented vertically and then later reoriented horizontally, controls that were filtered out due to the filterMisfits style are automatically filtered back into the tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
filterMisfits;	<i>Y</i>	<i>Y</i>	<i>N</i>

noDragDrop

Disables the dragging of a tool bar or tool bar buttons that reside within the tool bar and disables drops onto the tool bar or tool bar buttons.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noDragDrop;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

ISetCanvas		
border	deckOrientationId	packExpanded
bottomAlign	decksByGroup	packTight
centerAlign	horizontalDecks	rightAlign

IToolBar

ISetCanvas		
centerVerticalAlign	leftAlign	textId
classDefaultStyle	packEven	topAlign

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IToolBar contains the following nested classes:

IToolBar::WindowCursor (see page 630)
IToolBar::FrameCursor (see page 627)
IToolBar::Style (see page 629)

Location

```
Location {  
    aboveClient,    belowClient,    leftOfClient,    rightOfClient,    floating,  
    hidden  
};
```

Use these enumerators to specify the location of the tool bar.

IToolBar

aboveClient

The tool bar is above the client window.

belowClient

The tool bar is below the client window.

leftOfClient

The tool bar is to the left of the client window.

rightOfClient

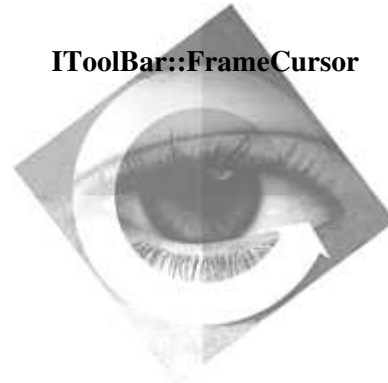
The tool bar is to the right of the client window.

floating

The tool bar is a child of an enclosing frame window that is a child of the desktop. If a user closes a floating tool bar, the location of the tool bar is set to hidden.

hidden

The tool bar is not displayed on the screen and is not visible to the user.



IToolBar::FrameCursor

Derivation IBase
IVBase
IToolBar::FrameCursor

Inherited By None.

Header File itbar.hpp

Members	Member	Page	Member	Page
	Constructor	627	setToFirst	628
	FrameCursor	627	setToNext	628
	invalidate	627	~FrameCursor	627
	isValid	628		

The IToolBar::FrameCursor class creates and manages a cursor for iterating over the tool bars that are owned by a frame window.

Public Functions

Constructors

You can construct and destruct objects of this class.

FrameCursor Creates an IToolBar::FrameCursor to iterate over the tool bars for the specified frame window.

FrameCursor(const IFrameWindow* frame);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

~FrameCursor

virtual ~FrameCursor();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

Cursor Functions

Use these members to iterate over the tool bars that are owned by a frame window.

invalidate Invalidates the cursor.

IToolBar::FrameCursor

	<pre>virtual void invalidate();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
isValid	Returns true if the cursor is valid.			
	<pre>virtual Boolean isValid() const;</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
setToFirst	Initializes the cursor to point to the first tool bar.			
	<pre>virtual Boolean setToFirst();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
setToNext	Advances the cursor to point to the next tool bar.			
	<pre>virtual Boolean setToNext();</pre>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IToolBar::Style

Derivation IBase
 IBitFlag
 IToolBar::Style

Inherited By None.

Header File itbar.hpp

The nested class IToolBar::Style provides a set of valid styles for the static member functions of the class IToolBar.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

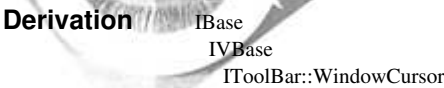
Inherited Protected Data

IBase		
recoverable	unrecoverable	



IToolBar::WindowCursor

IToolBar::WindowCursor



Inherited By None.

Header File itbar.hpp

Members					
	Member	Page	Member	Page	
	Constructor	630	setToNext	631	
	invalidate	631	WindowCursor	630	
	isValid	631	~WindowCursor	630	
	setToFirst	631			

The IToolBar::WindowCursor class creates and manages a cursor for iterating over the windows that have been added to a toolbar.

Public Functions

Constructors

You can construct and destruct objects of this class.

WindowCursor

Creates an IToolBar::WindowCursor to iterate over the windows that have been added to the specified tool bar.

WindowCursor(const IToolBar* toolbar);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

~WindowCursor

virtual ~WindowCursor();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	Y	Y	N

Cursor Functions

Use these members to iterate over the windows that have been added to a tool bar.

invalidate Invalidates the cursor.

virtual void invalidate();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

isValid Returns true if the cursor is valid.

virtual Boolean isValid() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToFirst Initializes the cursor to point to the first window in the tool bar.

virtual Boolean setToFirst();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setToNext Advances cursor to point to the next window in the tool bar.

virtual Boolean setToNext();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IToolBarButton

IToolBarButton

Derivation

IBase
 IVBase
 INotifier
 IWindow
 IControl
 ITextControl
 IButton
 ICustomButton
 IToolBarButton

Inherited By None.

Header File itbarbut.hpp

Members

Member	Page	Member	Page
Constructor	634	resetTransparentColor	639
allowsDragDelete	636	setBitmap	633
bitmap	633	setDefaultStyle	638
bitmapAndTextVisible	643	setDefaultTransparentColor	640
bitmapSize	633	setLatchedBitmap	634
bitmapVisible	643	setLayoutDistorted	634
calcMinimumSize	642	setStandardBitmapSize	637
classDefaultStyle	643	setStandardTextLines	637
clearDefaultTransparentColor	639	setStandardTextWidth	637
convertToGUIStyle	638	setTransparentColor	640
defaultStyle	638	setView	640
defaultTransparentColor	639	standardBitmapSize	637
disableDragDelete	636	standardFormat	644
enableDragDelete	636	standardTextLines	637
hasTransparentColor	639	standardTextWidth	637
isBitmapVisible	640	textVisible	644
isDefaultTransparentColorSet	639	transparentColor	640
isStandardFormat	636	useIdForBitmap	644
isTextVisible	640	useIdForText	644
latchedBitmap	633	view	641
noDragDelete	643	~IToolBarButton	635

The IToolBarButton class creates and manages the tool bar button control window. IToolBarButton provides support for displaying a bitmap, text, or bitmap and text on a button.

Public Functions

Bitmaps

Use these members to set and query the bitmaps that are displayed for a button. If a latched bitmap is not supplied, the default bitmap is also used for the latched state. If the tool bar button has a style of standardFormat and the bitmap is larger than the standard bitmap size, then the bitmap is resized to the standard bitmap size when setBitmap or setLatchedBitmap is used. You can also specify a transparent color for use when drawing the bitmap.

bitmap Returns the handle of the default bitmap.

IBitmapHandle bitmap() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

bitmapSize Returns the size of the current bitmap.

virtual ISize bitmapSize() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

latchedBitmap

Returns the handle of the bitmap that is displayed when the button is in the latched state.

IBitmapHandle latchedBitmap() const;	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setBitmap Sets the bitmap to be used for the unlatched (default) button state.

1 virtual IToolBarButton& setBitmap(const IResourceId& bitmapId);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

2 virtual IToolBarButton& setBitmap(const IBitmapHandle& handle);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

3 virtual IToolBarButton& setBitmap(unsigned long id);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBarButton

setLatchedBitmap

Sets the bitmap to be used for the latched state.

1	<code>virtual IToolBarButton& setLatchedBitmap(unsigned long id);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
2	<code>virtual IToolBarButton& setLatchedBitmap(const IResourceId& latchedBitmapId);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
3	<code>virtual IToolBarButton& setLatchedBitmap(const IBitmapHandle& handle);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>

Canvas Support

These members provide canvas support for tool bar buttons.

setLayoutDistorted

Indicates that changes have occurred in the window that will cause the layout of the window in a canvas to be updated. This implementation is used to optimize minimum size calculations.

<code>virtual IToolBarButton& setLayoutDistorted(unsigned long layoutAttributeOn, unsigned long layoutAttributeOff);</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
---	-------------------------------	------------------------------	---------------------------------

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IToolBarButton objects because both the copy constructor and assignment operator are private functions.

IToolBarButton

<code>IToolBarButton(unsigned long id, IWindow* parent, IWindow* owner, const IRectangle& initial = IRectangle (), const Style& style = defaultStyle ());</code>	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
--	-------------------------------	------------------------------	---------------------------------

Creates an IToolBarButton object from a control ID, parent and owner windows, rectangle, and style.

IToolBarButton

The file icconst.h includes defines for the following bitmaps and text that you can use when constructing tool bar buttons for standard operations:

- IC_ID_OPEN
- IC_ID_SAVE
- IC_ID_PRINT
- IC_ID_SETTINGS
- IC_ID_HELP
- IC_ID_CUT
- IC_ID_COPY
- IC_ID_PASTE
- IC_ID_COPYTO
- IC_ID_LOCATE
- IC_ID_BOLD
- IC_ID_ITALIC
- IC_ID_UNDERSCORE

The following figure shows the tool bar buttons provided.



Figure 4. Tool Bar Buttons

Exceptions	
InvalidParameter	The parent window pointer is invalid.

IToolBarButton

```
virtual
    ~IToolBarButton();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IToolBarButton

Shredder Support

Use these members to set and query whether the tool bar button can be dropped on a Workplace Shell shredder object.

allowsDragDelete

Returns true if the noDragDelete style is not set for the tool bar button.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
allowsDragDelete() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

disableDragDelete

Sets the noDragDelete style for the tool bar button to prevent it from being dropped on a Workplace Shell shredder object.

virtual IToolBarButton& disableDragDelete();	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

enableDragDelete

Removes the noDragDelete style for the tool bar button to enable it to be dropped on a Workplace Shell shredder object.

virtual IToolBarButton& enableDragDelete(Boolean enable = true);	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Standard Format

Use these members to set and query the values used for the standard formatting of tool bar buttons. Standard formatting controls the amount of area occupied by the bitmap (when visible) and the amount of area occupied by text (when visible). Standard formatting affects all of the tool bar buttons that have a style of IToolBarButton::standardFormat so that they have a consistent appearance.

isStandardFormat

Returns true if the button has a style of standardFormat.

Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isStandardFormat() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBarButton

setStandardBitmapSize

Sets the size of the area reserved for a bitmap for standard tool bar buttons when the bitmap is visible. The default size is 20 by 17.

```
static void  
    setStandardBitmapSize( const ISize& newSize = ISize ( 20 ,  
                                                            17 ) );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setStandardTextLines

Sets the number of lines to be displayed for standard tool bar buttons when text is visible. The default value is 1.

```
static void  
    setStandardTextLines( unsigned long newLines = 1 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setStandardTextWidth

Sets the width to be used for standard tool bar buttons when text is visible. The default value is 50.

```
static void  
    setStandardTextWidth( unsigned long newWidth = 50 );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

standardBitmapSize

Returns the size of the area reserved for a bitmap for a standard tool bar button when the button's bitmap is visible. The default size is 20 by 17.

```
static ISize  
    standardBitmapSize();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

standardTextLines

Returns the number of lines of text displayed for a standard tool bar button when the button's text is visible. The default value is 1.

```
static unsigned long  
    standardTextLines();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

standardTextWidth

Returns the width of a standard tool bar button when the text is visible. The default value is 50.

IToolBarButton

```
static unsigned long  
standardTextWidth();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Styles

Use these members to set and query IToolBarButton styles. You can use these styles with the styles defined by the following classes:

- IWindow (Vol. II)
- IControl (Vol. II)
- IButton (Vol. II)
- ICustomButton (p. 277)

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles, those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

```
virtual unsigned long  
convertToGUIStyle( const IBitFlag& style,  
                  Boolean extendedOnly = false ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

defaultStyle Returns the default style. The default style is classDefaultStyle unless you have changed the style using setDefaultStyle.

```
static Style  
defaultStyle();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultStyle

Sets the default style for all subsequent tool bar buttons.

style Use the styles provided by IToolBarButton::Style to specify the default style.

```
static void  
setDefaultStyle( const Style& style );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Transparency

Use these members to control transparency when drawing the bitmap. A transparent color is used to indicate part of the bitmap that should not be drawn for the bitmap. For example, the area under the bitmap is not overpainted for areas of the bitmap that are set to the transparent

IToolBarButton

color. These members do not actually modify the bitmap and are only used when the bitmap is drawn.

Transparency allows a bitmap to display a nonrectangular image with the correct button background color. Transparency support is important for tool bar buttons because the background color used for the latched state can be different from the default background color.

The default transparent color is IColor::pink (which has RGB values of red=255, green=0, blue=255).

clearDefaultTransparentColor

Resets the default transparent color so that no color will be treated as transparent for subsequent buttons.

static void	<u>Win</u>	<u>PM</u>	<u>Motif</u>
clearDefaultTransparentColor();	<i>Y</i>	<i>Y</i>	<i>N</i>

defaultTransparentColor

Returns the current default transparent color. This color is initially set to IColor::pink (which has RGB values of Red=255, Green=0, Blue=255).

static IColor	<u>Win</u>	<u>PM</u>	<u>Motif</u>
defaultTransparentColor();	<i>Y</i>	<i>Y</i>	<i>N</i>

hasTransparentColor

Returns true if a transparent color has been set.

virtual Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
hasTransparentColor() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

isDefaultTransparentColorSet

Returns true if a default transparent color is set.

static Boolean	<u>Win</u>	<u>PM</u>	<u>Motif</u>
isDefaultTransparentColorSet();	<i>Y</i>	<i>Y</i>	<i>N</i>

resetTransparentColor

Resets the button so that no color is made transparent when drawing the bitmap.

IToolBarButton

<code>virtual IToolBarButton& resetTransparentColor();</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultTransparentColor

Sets the default transparent color for all subsequent tool bar buttons.

<code>static void setDefaultTransparentColor(const IColor& aColor = IColor (IColor::pink));</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setTransparentColor

Sets the color to be made transparent when drawing the bitmap.

<code>virtual IToolBarButton& setTransparentColor(const IColor& color = IColor::pink);</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

transparentColor

Returns the current transparent color.

<code>virtual IColor transparentColor() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

Views

Use these members to set and query the current view of a tool bar button.

isBitmapVisible

Returns true if the bitmap is visible.

<code>Boolean isBitmapVisible() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

isTextVisible Returns true if the tool bar button text is visible.

<code>Boolean isTextVisible() const;</code>	<u>Win</u>	<u>PM</u>	<u>Motif</u>
	<i>Y</i>	<i>Y</i>	<i>N</i>

setView

Sets the current view of the tool bar button.

Note: The view of the tool bar button is changed to match the view of other tool bar buttons when it is added to a tool bar.

IToolBarButton

```
virtual IToolBarButton&  
    setView( View buttonView );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

view Returns the current view of the tool bar button.

```
View  
    view() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

ICustomButton		
backgroundColor	isAutoLatchEnabled	resetLatchedBackgroundColor
convertToGUIStyle	isLatched	resetLatchedForegroundColor
defaultStyle	isLatchedBackgroundColorHalftone	setDefaultStyle
disableAutoLatch	isLatchingEnabled	setLatchedBackgroundColor
disableLatching	latch	setLatchedForegroundColor
enableAutoLatch	latchedBackgroundColor	setUserData
enableLatching	latchedForegroundColor	unlatch

IButton		
allowsMouseClickedFocus	disableMouseClickedFocus	highlight
backgroundColor	enableMouseClickedFocus	hiliteBackgroundColor
click	enableNotification	hiliteForegroundColor
disabledForegroundColor	foregroundColor	isHighlighted

ITextControl		
clipboardHasTextFormat	setLayoutDistorted	text
displaySize	setText	textLength

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IToolBarButton

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Canvas Layout

These members support canvas layout for tool bar buttons.

calcMinimumSize

Returns the recommended minimum size of this tool bar button control. The size is based on the bitmap, button text and current font.

```
virtual ISize  
    calcMinimumSize() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Protected Functions

ICustomButton		
calcMinimumSize		

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these members to set and query IToolBarButton styles. You can use these styles with the styles defined by the following classes:

IToolBarButton

IWindow (Vol. II)
IControl (Vol. II)
IButton (Vol. II)
ICustomButton (p. 277)

bitmapAndTextVisible

Displays the bitmap and tool bar button text.

Note: The view of the tool bar button is changed to match the view of other tool bar buttons when it is added to a tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bitmapAndTextVisible;	<i>Y</i>	<i>Y</i>	<i>N</i>

bitmapVisible

Displays the bitmap.

Note: The view of the tool bar button is changed to match the view of other tool bar buttons when it is added to a tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
bitmapVisible;	<i>Y</i>	<i>Y</i>	<i>N</i>

classDefaultStyle

Specifies the original default style for this class, which is
IToolBarButton::bitmapVisible, IToolBarButton::useIdForBitmap,
IToolBarButton::useIdForText, IToolBarButton::standardFormat,
IButton::noPointerFocus, and IWindow::visible.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	<i>Y</i>	<i>Y</i>	<i>N</i>

noDragDelete

Prevents the tool bar button from being dropped on a Workplace Shell shredder object.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noDragDelete;	<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBarButton

standardFormat

Displays the tool bar button in a standard format for the application. All tool bar buttons that have this style are formatted the same (based on the standard bitmap size, text width, and number of text lines).

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
standardFormat;	<i>Y</i>	<i>Y</i>	<i>N</i>

textVisible

Displays the tool bar button text.

Note: The view of the tool bar button is changed to match the view of other tool bar buttons when it is added to a tool bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
textVisible;	<i>Y</i>	<i>Y</i>	<i>N</i>

useIdForBitmap

Uses the window ID of the tool bar button as the ID for the bitmap resource when the button is constructed.

If a bitmap resource cannot be found with the specified ID, the library tries to load a default bitmap for the tool bar button. If the default bitmap cannot be loaded, then the tool bar button does not use a bitmap.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
useIdForBitmap;	<i>Y</i>	<i>Y</i>	<i>N</i>

useIdForText

Uses the window ID of the tool bar button as the ID for the text resource when the button is constructed.

If a string resource cannot be found with the specified ID, no text is set for the button.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
useIdForText;	<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Data

ICustomButton		
autoLatch	classDefaultStyle	latchable

IToolBarButton

IButton		
buttonClickId	noPointerFocus	

ITextControl		
textId		

IControl		
group	tabStop	

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IToolBarButton contains the following nested classes:

IToolBarButton::Style (see page 647)

View

```
View {
    bitmapView,
    textView,
    bitmapAndTextView
};
```

IToolBarButton

Use these enumerations to specify the view of the button.

bitmapView

Only the bitmap is visible.

textView

Only the text is visible.

bitmapAndTextView

Both the bitmap and text are visible.

IToolBarButton::Style



IToolBarButton::Style

Derivation IBase
 IBitFlag
 IToolBarButton::Style

Inherited By None.

Header File itbarbut.hpp

The nested class IToolBarButton::Style provides a set of valid styles for the static member functions of the class IToolBarButton.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IToolBarContainer

IToolBarContainer

Derivation

IBase
IVBase
INotifier
IWindow
IControl
ICanvas
ISetCanvas
IToolBarContainer

Inherited By None.

Header File itbarcnr.hpp

Members	Member	Page	Member	Page
	Constructor	649	location	649
	classDefaultStyle	652	setDefaultStyle	650
	convertToGUIStyle	650	setLayoutDistorted	648
	defaultStyle	650	setLocation	649
	frameToolBarContainer	649	~IToolBarContainer	649

The IToolBarContainer class creates and manages a tool bar container for a frame window. Objects of this class are created by IToolBar objects when calling the IToolBar::createToolBarContainer function.

Public Functions

Canvas Layout

These members support canvas layout.

setLayoutDistorted

Treats a minimum size change for a child window like a layout change so that the container resizes as new windows are added or removed from tool bars.

```
virtual IToolBarContainer&
    setLayoutDistorted( unsigned long layoutAttributeOn,
                        unsigned long layoutAttributeOff );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	N

IToolBarContainer

Constructors

You can construct and destruct objects of the this class. You cannot copy or assign IToolBarContainer objects because both the copy constructor and assignment operator are private functions.

IToolBarContainer

IToolBarContainer(unsigned long identifier,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
IFrameWindow* parentAndOwner,	<i>Y</i>	<i>Y</i>	<i>N</i>
IFrameWindow::Location location,			
const Style& style = defaultStyle ());			

Creates an IToolBarContainer object by providing a window ID, frame window, location, and style.

~IToolBarContainer

virtual	<u>Win</u>	<u>PM</u>	<u>Motif</u>
~IToolBarContainer();	<i>Y</i>	<i>Y</i>	<i>N</i>

Frame Location

Use these members to set and query the tool bar container's location.

frameToolBarContainer

Returns the tool bar container for the specified frame and location.

static IToolBarContainer*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
frameToolBarContainer(<i>Y</i>	<i>Y</i>	<i>N</i>
IFrameWindow* frame,			
IFrameWindow::Location frameLocation);			

location Returns the current frame window location of the tool bar container.

IFrameWindow::Location	<u>Win</u>	<u>PM</u>	<u>Motif</u>
location() const;	<i>Y</i>	<i>Y</i>	<i>N</i>

setLocation Sets a new frame window location for the tool bar container.

virtual IToolBarContainer&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
setLocation(IFrameWindow::Location location);	<i>Y</i>	<i>Y</i>	<i>N</i>

IToolBarContainer

Styles

Use these members to set and query IToolBarContainer styles. You can use these styles with the styles defined by the following nested classes:

IWindow Styles (Vol. II)
ICanvas Styles (p. 35)
ISetCanvas Styles (p. 574)

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles, those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

```
virtual unsigned long  
    convertToGUIStyle( const IBitFlag& style,  
                      Boolean extendedOnly = false ) const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

defaultStyle Returns the default style. The default style is classDefaultStyle unless you have changed the style using setDefaultStyle.

```
static Style  
    defaultStyle();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

setDefaultStyle

Sets the default style for all subsequent tool bar containers.

style Use the styles provided by IToolBarContainer::Style to specify the default style.

```
static void  
    setDefaultStyle( const Style& style );
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>N</i>

Inherited Public Functions

ISetCanvas		
alignment	packType	setLayoutDistorted
convertToGUIStyle	pad	setMargin
deckCount	position	setPackType
deckOrientation	setAlignment	setPad

IToolBarContainer

ISetCanvas		
defaultStyle	setDeckCount	setText
groupPad	setDeckOrientation	size
margin	setDefaultStyle	text
moveSizeTo	setGroupPad	topHandle

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

ISetCanvas		
expandForText	layout	

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks

IToolBarContainer

ICanvas		
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Styles

Use these members to set and query IToolBarContainer styles. You can use these styles with the styles defined by the following nested classes:

- IWindow Styles (Vol. II)
- ICanvas Styles (p. 35)
- ISetCanvas Styles (p. 574)

classDefaultStyle

Specifies the original default style for this class, which is IWindow::visible.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	N

Inherited Public Data

ISetCanvas		
border	deckOrientationId	packExpanded
bottomAlign	decksByGroup	packTight
centerAlign	horizontalDecks	rightAlign
centerVerticalAlign	leftAlign	textId
classDefaultStyle	packEven	topAlign

ICanvas		
classDefaultStyle		

IToolBarContainer

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IToolBarContainer contains the following nested classes:

IToolBarContainer::Style (see page 654)



IToolBarContainer::Style

IToolBarContainer::Style



Inherited By None.

Header File itbarcnr.hpp

The nested class IToolBarContainer::Style provides a set of valid styles for the static member functions of the class IToolBarContainer.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	



IToolBarFrameWindow

Derivation

```

IBase
  IVBase
    INotifier
      IWindow
        IFrameWindow
          IToolBarFrameWindow
  
```

Inherited By None.

Header File itbarfrm.hpp

Members	Member	Page	Member	Page
	Constructor	656	titleText	657
	convertToGUIStyle	656	~IToolBarFrameWindow	656
	setTitleText	656		

The IToolBarFrameWindow class is used to contain floating tool bars.

IToolBarFrameWindows are created when a user constructs a tool bar with a location of floating or calls IToolBar::setLocation with a value of IToolBar::floating.

PM IToolBarFrameWindows are visually distinct from other frame windows in that the title bar height of IToolBarFrameWindows is smaller than other frame windows. In addition, there are two buttons that appear where the frame window's minimize and maximize buttons normally are.

The left button is a toggle button that allows you to pin or attach a tool bar frame window to its owning frame window. This has the effect that the tool bar stays in the same location relative to its owner when the owner is moved.

The right button allows you to expand and collapse the tool bar contained in the tool bar frame window. This allows you to shrink the tool bar to recover screen area when you are not actively using the tool bar and later expand the tool bar to show all tool bar controls.

Win The maximize/restore button on an IToolBarFrameWindow allows you to expand and collapse the tool bar contained in the tool bar frame window. This allows you to shrink the tool bar to recover screen area when you are not actively using the tool bar and later expand the tool bar to show all tool bar controls.

IToolBarFrameWindow

Public Functions

Constructors

You can construct and destruct objects of this class.

IToolBarFrameWindow

IToolBarFrameWindow(unsigned long windowId, IFrameWindow* owner);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
--	-----------------	----------------	-------------------

Creates and IToolBarFrameWindow object with the specified window ID and the specified frame window as owner.

~IToolBarFrameWindow

virtual ~IToolBarFrameWindow();	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
------------------------------------	-----------------	----------------	-------------------

Styles

Use this function to convert style bits into a format recognized by the GUI.

convertToGUIStyle

Use this function to convert style bits into the style value that can be processed by the GUI. The default action is to return the base GUI style for the platform. Extended styles , those defined by the User Interface Class Library, can be returned by setting the *extendedOnly* parameter to true.

virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

Title Text

Use these members to set or query the title text of an IToolBarFloatingFrame object.

setTitleText Sets the text of the tool bar frame's title bar.

 virtual IToolBarFrameWindow& setTitleText(const char* text);	<u>Win</u> Y	<u>PM</u> Y	<u>Motif</u> N
---	-----------------	----------------	-------------------

IToolBarFrameWindow

2	virtual IToolBarFrameWindow& setTitleText(const IResourceId& text);	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
----------	--	-------------------------------	------------------------------	---------------------------------

titleText Returns the text of the tool bar frame's title bar.

IString titleText() const;	<u>Win</u> <i>Y</i>	<u>PM</u> <i>Y</i>	<u>Motif</u> <i>N</i>
-------------------------------	-------------------------------	------------------------------	---------------------------------

Inherited Public Functions

IFrameWindow		
addExtension	isAnExtension	setBorderSize
addToWindowList	isFlashing	setBorderWidth
backgroundColor	isMaximized	setClient
beginFlashing	isMinimized	setDefaultOrdering
borderHeight	isModal	setDefaultStyle
borderSize	matchForMnemonic	setDestroyOnClose
borderWidth	maximize	setExtensionSize
client	maximizeRect	setIcon
clientHandle	minimize	setLayoutDistorted
clientRectFor	minimizeRect	setMousePointer
close	mousePointer	setRestoreRect
convertToGUIStyle	moveSizeToClient	setResult
defaultOrdering	nextShellRect	setToolBarList
defaultPushButton	notifyOwner	shareParentDBCSSStatus
defaultStyle	removeExtension	show
disabledBackgroundColor	removeFromWindowList	showModally
dismiss	resetBackgroundColor	start
enableNotification	resetDisabledBackgroundColor	toolBarList
endFlashing	restore	topHandle
frameRectFor	restoreRect	update
handleFor	result	useExtensionMinimumSize
icon	setBorderHeight	usesDialogBackground

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IToolBarFrameWindow

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Functions

IFrameWindow		
addDefaultHandler	findExtensions	registerFrameClass
attachClient	initialize	removeDefaultHandler
create	isFrameWindow	setExtensions
extensions	isRelatedHandle	tryToLoadDialog
findExtension	registerCallbacks	unregisterCallbacks

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Inherited Public Data

IFrameWindow		
accelerator	deferCreation	minimized
activateId	dialogBackground	minimizedIcon
alignNoAdjust	dialogBorder	noMoveWithOwner
animated	hideButton	shellPosition
appDBCSStatus	horizontalScroll	sizingBorder
border	maximizeButton	systemMenu
classDefaultStyle	maximized	systemModal
closeId	menuBar	titleBar
deactivateId	minimizeButton	verticalScroll

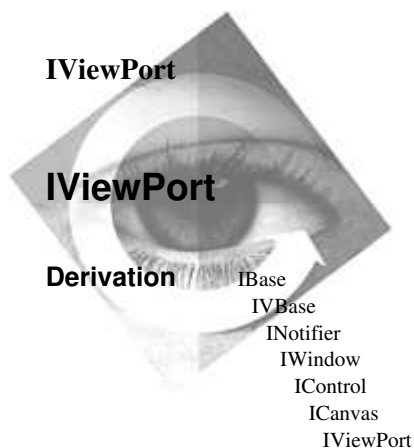
IToolBarFrameWindow

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	



Inherited By None.

Header File ivport.hpp

Members				
	Member	Page	Member	Page
	Constructor	662	positionViewWindow	667
	alwaysHorizontalScrollBar	669	scrollViewHorizontallyTo	665
	alwaysVerticalScrollBar	670	scrollViewVerticallyTo	665
	asNeededHorizontalScrollBar	670	setDefaultStyle	665
	asNeededVerticalScrollBar	670	setLayoutDistorted	664
	calcMinimumSize	667	setTopLeftViewPoint	668
	classDefaultStyle	670	setupScrollBars	668
	convertToGUIStyle	664	setViewWindowSize	665
	defaultStyle	664	topLeftViewPoint	668
	expandableViewWindow	670	verticalScrollBar	662
	horizontalScrollBar	662	viewWindow	666
	layout	667	viewWindowDrawRectangle	666
	noHorizontalScrollBar	671	viewWindowSize	666
	noVerticalScrollBar	671	viewWindowSizeId	669
	noViewWindowFill	671	~IViewport	663

The IViewport class provides scrolling support for another window. The window being scrolled is termed 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 (p. 26)
- ISetCanvas (p. 565)
- IMultiCellCanvas (p. 470)
- INotebook (p. 489)
- IContainerControl (p. 174)

IViewPort

Note: If you are using an IMultiCellCanvas with expandable rows or columns as the view window, specify the IViewPort::expandableViewWindow (p. 669) style when you construct the view port.

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 window identifier (ID) of the scrollable rectangular window is IC_VIEWPORT_VIEWRECTANGLE. If you enable the view window for direct manipulation, the direct manipulation events are dispatched to the scrollable rectangular 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 (p. 665) 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. If you use the IViewPort::expandableViewWindow style and no logical size, IViewPort can grow the view window larger than 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.



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 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 IViewPort::scrollViewHorizontallyTo (p. 665) and IViewPort::scrollViewVerticallyTo (p. 665).

IViewPort

Public Functions

Child Scroll Bars

Use these members to access the scroll bar windows created by the IViewPort window.

horizontalScrollBar

Returns the address of the horizontal scroll bar.

IScrollBar*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
horizontalScrollBar() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

verticalScrollBar

Returns the address of the vertical scroll bar.

IScrollBar*	<u>Win</u>	<u>PM</u>	<u>Motif</u>
verticalScrollBar() const;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Constructors

You can construct and destruct objects of this class. You cannot copy or assign IViewPort objects because both the copy constructor and assignment operator are private functions.

IViewPort

IViewPort(unsigned long windowIdentifier,	<u>Win</u>	<u>PM</u>	<u>Motif</u>
IWindow* parent,	<i>Y</i>	<i>Y</i>	<i>Y</i>
IWindow* owner,			
const IRectangle& initialSize = IRectangle (),			
const Style& style = defaultStyle ());			

windowIdentifier

The window identifier of the view port you are constructing.

We recommend that you do the following:

- For portability, use a value in the range 0 to 65535.
- Give unique identifiers to all windows in the same frame window. Two windows are in the same frame window if the first frame window in each of its parent window chains is the same window.
- Give the client window a window identifier of IC_FRAME_CLIENT_ID, which is defined in the file icconst.h.

IViewport

Presentation Manager Notes: Do not use FID_XXX values defined in pmwin.h, other than FID_CLIENT (which is equivalent to IC_FRAME_CLIENT_ID).

The OS/2 and Windows versions of the IViewport class reserve the following window identifiers for child windows that it creates:

IC_VIEWPORT_VERTSCROLLBAR

Vertical scroll bar.

IC_VIEWPORT_HORZSCROLLBAR

Horizontal scroll bar.

IC_VIEWPORT_VIEWRECTANGLE

Scroll rectangle window. 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 view port you are constructing. You must specify a parent window. The parent window is primarily used for visible relationships.
- owner* The owner window of the view port you are constructing. The owner window is primarily used for routing notification events and unprocessed messages. A window also inherits colors from its owner window.
- Motif Notes:** The owner window is only used for routing unprocessed messages. There is no concept of an owner in Motif.
- initialSize* The initial position and size of the view port you are constructing. The position is relative to the origin of the parent window. See ICoordinateSystem (Vol. II) for additional information. Optional.
- style* The view port's characteristics. This value can be a combination of IViewport::Style (p. 669), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects. Optional.

Exceptions	
InvalidParameter	<i>style</i> contains an invalid combination of styles. <i>style</i> cannot contain more than one horizontal scroll bar style (IViewport::alwaysHorizontalScrollBar, IViewport::asNeededHorizontalScrollBar, and IViewport::noHorizontalScrollBar), or more than one vertical scroll bar style (IViewport::alwaysVerticalScrollBar, IViewport::asNeededVerticalScrollBar, and IViewport::noVerticalScrollBar).

~IViewport

IViewport

<pre>virtual ~IViewport();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

setLayoutDistorted

Processes a size change as a layout change.

<pre>virtual IViewport& setLayoutDistorted(unsigned long layoutAttributeOn, unsigned long layoutAttributeOff);</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Styles

Use these members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

ICanvas::Style (p. 35)
IWindow::Style (Vol. II)

Once you have constructed an IViewport object, you can use IViewport, ICanvas, and IWindow member functions to query and change individual styles.

convertToGUIStyle

Converts a style object into a value appropriate for the presentation system. The default action is to return the base GUI styles for the platform. Extended styles, those defined by the application and the User Interface Class Library, will be returned if you set *extendedOnly* to true.

<pre>virtual unsigned long convertToGUIStyle(const IBitFlag& style, Boolean extendedOnly = false) const;</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>N</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>N</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>N</i>					

defaultStyle Returns the default style. The default style is classDefaultStyle (p. 670) unless you change the style using setDefaultStyle (p. 665).

<pre>static Style defaultStyle();</pre>	<table><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

IViewPort

setDefaultStyle

Sets the default style for all subsequent view ports.

static void		<u>Win</u>	<u>PM</u>	<u>Motif</u>
setDefaultStyle(const Style& style);		<i>Y</i>	<i>Y</i>	<i>Y</i>

style A combination of IViewPort::Style (p. 669), ICanvas::Style (p. 35), and IWindow::Style (Vol. II) objects.

View Window

Use these members to change the appearance of the view window in the view port.

scrollViewHorizontallyTo

Scrolls the view window horizontally.

virtual IViewPort&		<u>Win</u>	<u>PM</u>	<u>Motif</u>
scrollViewHorizontallyTo(unsigned long leftOffset);		<i>Y</i>	<i>Y</i>	<i>Y</i>

leftOffset The 0-based offset from the left edge of the view window to scroll to the left edge of the view port.

scrollViewVerticallyTo

Scrolls the view window vertically.

virtual IViewPort&		<u>Win</u>	<u>PM</u>	<u>Motif</u>
scrollViewVerticallyTo(unsigned long topOffset);		<i>Y</i>	<i>Y</i>	<i>Y</i>

topOffset The 0-based offset from the top edge of the view window to scroll to the top edge of the view port.

setViewWindowSize

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.

virtual IViewPort&		<u>Win</u>	<u>PM</u>	<u>Motif</u>
setViewWindowSize(const ISize& size);		<i>Y</i>	<i>Y</i>	<i>Y</i>

IViewport

size New logical size of the view window.

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.

```
virtual IWindowHandle  
viewWindow();
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Exceptions	
InvalidRequest	The view port has found more than one view window. A view port can have only one child window.

viewWindowDrawRectangle

Returns the area of the view window that is currently visible in the view port.

```
virtual IRectangle  
viewWindowDrawRectangle() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

viewWindowSize

Returns the logical size of the window being scrolled. If you have not called IViewport::setViewWindowSize (p. 665) to set a logical size, the real size of the view window is returned.

```
virtual ISize  
viewWindowSize() const;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Functions

ICanvas		
backgroundColor	defaultStyle	origDefaultButtonHandle
convertToGUIStyle	isTabStop	setDefaultStyle
defaultPushButton	matchForMnemonic	setFont

IControl		
disableGroup	enableGroup	isGroup
disableTabStop	enableTabStop	isTabStop

IViewPort

INotifier		
disableNotification	enableNotification	isEnabledForNotification

IVBase		
asDebugInfo	asString	

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Note: Members of IWindow (Vol. II) are not shown.

Protected Functions

Layout Support

Layout members determine how this class sizes and positions its child windows or how this window will be laid out on another canvas.

calcMinimumSize

Returns the recommended minimum size for this IViewPort object. The size is based on the minimum size of the vertical and horizontal scroll bars.

virtual ISize	<u>Win</u>	<u>PM</u>	<u>Motif</u>
calcMinimumSize() const;	Y	Y	Y

layout

Positions the scroll bars, scroll rectangle window, and view window.

virtual IViewPort&	<u>Win</u>	<u>PM</u>	<u>Motif</u>
layout();	Y	Y	Y

View Window

Use these members to change the appearance of the view window in the view port.

positionViewWindow

Positions the view window within the view rectangle.

IViewPort

<pre>virtual IViewPort& positionViewWindow(const IWindowHandle& viewWindow, const IRectangle& viewRectangle);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

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

Sets the view window coordinate that displays in the upper-left corner of the view rectangle.

<pre>virtual IViewPort& setTopLeftViewPoint(const IPoint& topLeft);</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

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` (p. 665) or `scrollViewVerticallyTo` (p. 665) instead of `setTopLeftViewPoint`. This function does not update the scroll bars.

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` (p. 667).

<pre>virtual IViewPort& setupScrollBars();</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

topLeftViewPoint

Returns the view window coordinate that displays in the upper-left corner of the view rectangle.

<pre>IPoint topLeftViewPoint() const;</pre>	<table border="0"><tr><td><u>Win</u></td><td><u>PM</u></td><td><u>Motif</u></td></tr><tr><td><i>Y</i></td><td><i>Y</i></td><td><i>Y</i></td></tr></table>	<u>Win</u>	<u>PM</u>	<u>Motif</u>	<i>Y</i>	<i>Y</i>	<i>Y</i>
<u>Win</u>	<u>PM</u>	<u>Motif</u>					
<i>Y</i>	<i>Y</i>	<i>Y</i>					

Inherited Protected Functions

ICanvas		
areChildrenReversed	initialize	passEventToOwner
calcMinimumSize	layout	registerCallbacks
fixupChildren	layoutSize	setLayoutSize

INotifier		
addObserver	notifyObservers	observerList

Note: Members of IWindow (Vol. II) are not shown.

Public Data

Notification Members

These members allow observers to process changes to the window.

viewWindowSizeId

Notification identifier provided to observers when the logical size of the view port's view window changes. IViewPort provides a pointer to an ISize object for the new size in the eventData field of the INotificationEvent.

```
static INotificationId const
viewWindowSizeId;
```

<u>Win</u>	<u>PM</u>	<u>Motif</u>
Y	Y	Y

Styles

Use these members to customize a window at the time you construct it. Most styles have equivalent member functions, which allow you to similarly modify a window after creating it. You can use these styles with the styles defined by the following nested classes:

```
ICanvas::Style (p. 35)
IWindow::Style (Vol. II)
```

Once you have constructed an IViewPort object, you can use IViewPort, ICanvas, and IWindow member functions to query and change individual styles.

alwaysHorizontalScrollBar

Specifies that the view port never removes the horizontal scroll bar.

IViewPort

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
alwaysHorizontalScrollBar;	Y	Y	Y

alwaysVerticalScrollBar

Specifies that the view port never removes the vertical scroll bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
alwaysVerticalScrollBar;	Y	Y	Y

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.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
asNeededHorizontalScrollBar;	Y	Y	Y

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.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
asNeededVerticalScrollBar;	Y	Y	Y

classDefaultStyle

Specifies the original default style for this class, which is
IViewPort::asNeededHorizontalScrollBar | IViewPort::asNeededVerticalScrollBar |
IWindow::visible.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
classDefaultStyle;	Y	Y	Y

expandableViewWindow

Specifies that the view port will size its view window to completely fill the view port by sizing the view window to the larger of the following:

- The minimum size of the view window
- The size of the view rectangle.

When the view port sizes the view window to its minimum size, it allows the user to scroll the view window if it is not completely visible.

IViewPort

Use this style when using an IMultiCellCanvas (p. 470) with expandable rows or columns as the view window.

Note: Setting a logical view window size through the function IViewPort::setViewWindowSize (p. 665) causes this style to be ignored.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
expandableViewWindow;	<i>Y</i>	<i>Y</i>	<i>N</i>

noHorizontalScrollBar

Specifies that the view port never shows the horizontal scroll bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noHorizontalScrollBar;	<i>Y</i>	<i>Y</i>	<i>Y</i>

noVerticalScrollBar

Specifies that the view port never shows the vertical scroll bar.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noVerticalScrollBar;	<i>Y</i>	<i>Y</i>	<i>Y</i>

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.

static const Style	<u>Win</u>	<u>PM</u>	<u>Motif</u>
noViewWindowFill;	<i>Y</i>	<i>Y</i>	<i>Y</i>

Inherited Public Data

ICanvas		
classDefaultStyle		

IWindow		
activeColorId	disabledBackgroundColorId	noStyle
backgroundColorId	disabledForegroundColorId	positionId

IViewPort

IWindow		
borderColorId	enableId	saveBits
clipChildren	focusId	shadowColorId
clipSiblings	fontId	sizeId
clipToParent	foregroundColorId	synchPaint
commandId	hiliteBackgroundColorId	systemCommandId
deleteId	hiliteForegroundColorId	visible
disabled	inactiveColorId	visibleId

Note: Members of IWindow (Vol. II) are not shown.

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Nested Classes

IViewPort contains the following nested classes:

IViewPort::Style (see page 673)



IViewPort::Style

Derivation IBase
 IBitFlag
 IViewPort::Style

Inherited By None.

Header File ivport.hpp

The nested class IViewPort::Style provides a valid set of view port styles for the IViewPort::defaultStyle (p. 664) and IViewPort::setDefaultStyle (p. 665) functions, and for the constructor of the class IViewPort (p. 660). You can use these styles with the styles defined by the following nested classes:

 ICanvas::Style (p. 35)
 IWindow::Style (Vol. II)

Once you have constructed an IViewPort object, you can use IViewPort, ICanvas, and IWindow member functions to query and change individual styles.



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.

Inherited Public Functions

IBitFlag		
asExtendedUnsignedLong	asUnsignedLong	operator !=

IBase		
asDebugInfo	messageFile	setMessageFile
asString	messageText	version

Inherited Protected Functions

IViewPort::Style

IBitFlag		
setValue		

Inherited Protected Data

IBase		
recoverable	unrecoverable	

Glossary

This glossary defines terms and abbreviations that are used in this book. If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*, New York:McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

A

abstract class. (1) A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class. See also base class. (2) A class that allows polymorphism.

abstract data type. A mathematical model that includes a structure for storing data and operations that can be performed on that data. Common abstract data types include sets, trees, and heaps.

abstraction (data). See data abstraction.

access. An attribute that determines whether or not a class member is accessible in an expression or declaration. It can be public, protected, or private.

access declaration. A declaration used to adjust access to members of a base class.

access function. A function that returns information about the elements of an object so that you can analyze various elements of a string.

access resolution. The process by which the accessibility of a particular class member is determined.

access specifier. One of the C++ keywords public, private, or protected.

ambiguous derivation. A derivation where the class is derived from two or more base classes that have members with the same name.

amplifier. A device that increases the strength of input signals. Also referred to as an amp.

amplifier-mixer. A combination amplifier and mixer that is used to control the characters of an audio signal from one or more audio sources. Also referred to as an amp-mixer.

animate. Make or design in such a way as to create apparently spontaneous, lifelike movement.

animation rate. The number of thousandths of a second that pass before the next bitmap is displayed for a button while it is animated.

anonymous union. A union that is declared within a structure or class and that does not have a name.

area. In computer graphics, a filled shape, such as a solid rectangle.

array. An aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting.

array implementation. (In Collection Class Library) Implementation of an abstract data type using an array. Also called a tabular implementation.

ASCII (American National Standard Code for Information Interchange). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

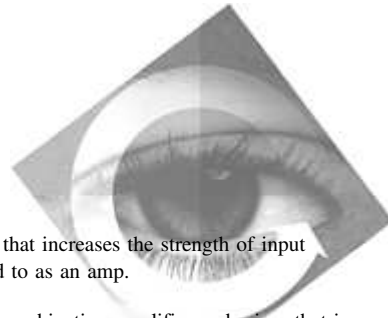
Note: IBM has defined an extension to ASCII code (characters 128-255).

audio. Pertaining to the portion of recorded information that can be heard.

audio attributes. The standard audio attributes are: mute, volume, balance, treble, and bass.

audio formats. The way the audio information is stored and interpreted.

audio track. (1) The audio (sound) portion of the program. (2) The physical location where the audio is placed beside the image. (A system with two sound tracks can have either



automatic storage •CD-XA

stereo sound or two independent sound tracks.) Synonymous with sound track.

automatic storage. Storage that is allocated on entry to a routine or block and is freed on the subsequent return. Sometimes referred to as *stack storage* or *dynamic storage*.

automatic storage management. The process that automatically allocates and deallocates objects in order to use memory efficiently.

auto-reset event. In Windows, an event used to signal a single thread that an application has completed.

See also event.

auxiliary classes. Classes that support other classes. Auxilliary classes in the Collection Class Library include classes for cursors, pointers and iterators.

AVL tree. A balanced binary search tree that does not allow the height of two siblings to differ by more than one.

B

B*-tree (B star tree). A tree in which only the leaves contain whole elements. All other nodes contain keys.

background color. The color in which the background of a graphic primitive is drawn.

balance. (1) For audio, refers to the relative strength of the left and right channels. A balance level of 0 is left channel only. A balance level of 100 is right channel only (2) A state of equilibrium, usually between treble and bass.

base class. A class from which other classes are derived. A base class may itself be derived from another base class. See also abstract class.

based on. A relationship between two classes in which one class is implemented through the other. A new class is “based on” an existing class when the existing class is used to implement it.

bass. The lower half of the whole vocal or instrumental tonal range.

bit field. A member of a structure or union that contains a specified number of bits.

bit mask. A pattern of characters used to control the retention or elimination of portions of another patterns of characters.

bits-per-sample. The number of bits of audio data that is to represent each sample of each channel (right or left). This is the resolution of the audio data. CD quality needs to be 16 bits-per-sample.

boundary alignment. The position in main storage of a fixed-length field (such as byte or doubleword) on an integral boundary for that unit of information.

For the Class Library example, a word boundary is a storage address evenly divisible by two.

bounded collection. A collection that has an upper limit on the number of elements it can contain.

brightness. The level of luminosity of the video signal. A brightness level of 0 produces a maximally white signal. A brightness level of 100 produces a maximally black signal.

built-in. A function that the compiler automatically puts inline instead of generating a call to the function.

C

camcorder. A compact, hand-held video camera with integrated videotape recorder.

canvas. Canvases are windows with a layout algorithm that manage child windows. The canvas classes are a set of window classes which allow you to implement dialog-like windows (that is, a window with several child controls). These windows are used for showing views of objects as both pages in a notebook and as windows that gather information to run an action. The different canvases can manage the size and position of child windows, provide moveable split bars between windows, and support the ability to scroll a window.

The canvases include the base class, ICanvas, and its four derived classes: IMultiCellCanvas, ISetCanvas, ISplitCanvas, and IViewport.

cast. A notation used to express the conversion of one type to another.

catch block. A block associated with a try block that receives control when a C++ exception matching its argument is thrown.

CD. Compact disc

CD-ROM. Compact disc-read-only memory

CD-XA. Compact disc-extended architecture

channel mapping •Compound Document Framework

channel mapping. The translation of a MIDI channel number for a sending device to an appropriate channel for a receiving device.

character array. An array of type char.

child. A node that is subordinate to another node in a tree structure. Only the root node of a tree is not a child.

child class. See derived class.

child window. A window derived from another window and drawn relative to it.

circular slider control. A 360-degree knob-like control that simulates the buttons on a TV, a stereo, or video components. By rotating the slider arm, the user can set, display, or modify a value, such as the balance, bass, volume, or treble.

class. A user-defined type. Classes can be defined hierarchically, allowing one class to be an expansion of another, and classes can restrict access to their members.

class hierarchy. A tree-like structure showing relationships among classes. It places one abstract class at the top (a base class) and one or more layers of derived classes below it.

class library. A collection of classes.

class template. A blueprint describing how a set of related classes can be constructed.

client area window. An intermediate window between an IFrameWindow and its controls and other child windows.

client program. A program that uses a class. The program is said to be a client of the class.

CLSID. The globally unique identifier for an object. The system Registry uses the CLSID to distinguish all OLE objects available on a system. A CLSID contains 32 hex digits.

collection. (1) In a general sense, an implementation of an abstract data type for storing elements. (2) An abstract class without any ordering, element properties, or key properties. All abstract Collection Classes are derived from Collection.

Collection Classes. A set of classes that implement abstract data types for storing elements.

color palette. A set of all the colors that can be used in a displayed image.

common controls. In Windows, a DLL that includes the following: a header control (a window for displaying

multiple columns of data), list view (a way to display objects as icons with labels), progress bar, property sheet, status bar, tool bar, track bar (slider control), tree view (an outline-type list), and an up-down control (spin control).

compact disc (CD). (1) A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser. (2) A disc with information stored in the form of pits along a spiral track. The information is decoded by a compact-disc player and interpreted as digital audio data, which most computers can process.

compact disc-extended architecture (CD-EX). A storage format that accommodates interleaved storage of audio, video, and standard file system data.

compact disc-read-only memory (CD-ROM). (1) An optical storage medium (2) High-capacity, read-only memory in the form of an optically read compact disc.

Complex Mathematics library. A C++ class library that provides the facilities to manipulate complex numbers and perform standard mathematical operations on them.

component. The unit of exchange in terms of OLE compound documents; also referred to as a document component. Components created using the Compound Document Framework are either container or server components.

component stationery. The “glue” that holds all pieces of a Compound Document Framework application together, including the model, view, and frame window. The template subclass IComponentStationeryFor, which takes a model and view as arguments, does the work of instantiating a component stationery. Synonym for stationery.

composite. The combination of two or more film, video, or electronic images into a single frame or display.

compound document. A single centralized location for integrating arbitrary or unstructured data from different sources. The Compound Document Framework provides a mechanism for creating document components by providing a structure that you can easily extend to create servers or containers. The framework stores compound documents using the OLE structured storage specification (that is, docfiles).

Compound Document Framework. A starting point for creating a server or container document component that is OLE-enabled. Compound documents allow for the integration of arbitrary or unstructured data from different sources into one centralized location.

computer-controlled device •degree

computer-controlled device. An external video source device with frame-stepping capability, usually a videodisc player, whose output can be controlled by the multimedia subsystem.

concrete class. A class that implements an abstract data type but does not allow polymorphism.

const. (1) An attribute of a data object that declares that the object cannot be changed. (2) An attribute of a function that declares that the function will not modify data members of its class.

constructor. A special class member function that has the same name as the class and is used to construct and possibly initialize objects of its class type. A return type is not specified.

container. A holder for zero or more embedded components. Containers manage the compound document by maintaining a list of embedded components and storing the container and its embedded component's data when requested. Containers built with the Compound Document Framework are also servers and can therefore be embedded inside of other containers.

containment function. A function that determines whether a collection contains a given element.

control. A graphic object that represents operations or properties of other objects.

See also tree control.

copy constructor. A constructor used to make a copy of an object from another object of the same type.

critical section. (1) Code that must be executed by one thread while all other threads in the process are suspended. (2) In Windows, a synchronization object. A critical section is not a kernel object; that is, it is not managed by the low-level components of the operating system and is not manipulated using handles. (3) In Windows, a small section of code that requires exclusive access to some shared data before the code can execute. Critical threads synchronize threads only within a single process, and they allow only one thread at a time to gain access to a region of data.

See also mutex, semaphore, and event. Contrast with kernel object.

cursor. A reference to an element at a specific position in a data structure.

cursor iteration. The process of repeatedly moving the cursor to the next element in a collection until some condition is satisfied.

cursor emphasis. When the selection cursor is on a choice, that choice has cursored emphasis.

C/2. A version of the C language designed for the OS/2 environment.

D

daemon. A program that runs unattended to perform a service for other programs.

data abstraction. A data type with a private representation and a public set of operations. The C++ language uses the concept of classes to implement data abstraction.

DBCS (Double-Byte Character Set). See double-byte character set.

deck. A line of child windows in a set canvas that is direction-independent. A horizontal deck is equivalent to a row and a vertical deck is equivalent to a column.

declaration. Introduces a name to a program and specifies how the name is to be interpreted.

declare. To specify the interpretation that C++ gives to each identifier.

default argument. An argument that is declared with a default value in a function prototype or declaration. If a call to the function omits this argument, the default value is used. Arguments with default values must be the trailing arguments in a function prototype argument list.

default class. A class with preprogrammed definitions that can be used for simple implementations.

default constructor. A constructor that takes no arguments, or a constructor for which all the arguments have default values.

default implementation. One of several possible implementation variants offered as the default for a specific abstract data type.

default operation class. A class with preprogrammed definitions for all required element and key operations for a particular implementation.

degree. The number of children of a node.

delete •dynamic casting

delete. (1) A C++ keyword that identifies a free-storage deallocation operator. (2) A C++ operator used to destroy objects created by operator new.

deque. A queue that can have elements added and removed at both ends. A double-ended queue.

dequeue. An operation that removes the first element of a queue.

derivation. (1) The creation of a new or derived class from an existing base class. (2) The relationship between a class and the classes above or below it in a class hierarchy.

derived class. A class that inherits from a base class. You can add new data members and member functions to the derived class. You can manipulate a derived class object as if it were a base class object. The derived class can override virtual functions of the base class.

Synonym for child class and subclass.

destructor. A special member function that has the same name as its class, preceded by a tilde (~), and that “cleans up” after an object of that class, for example, by freeing storage that was allocated when the object was created. A destructor has no arguments, and no return type is specified.

difference. Given two sets A and B, the difference (A-B) is the set of all elements contained in A but not in B.

digital audio. Audio data that has been converted to digital form.

digital video. Material that can be seen and that has been converted to digital form.

digital video device. A full-motion video device that can record or play files (or both) containing digitally stored video.

diluted array. An array in which elements are deleted by being flagged as deleted, rather than by actually removing them from the array and shifting later elements to the left.

diluted sequence. A sequence implemented using a diluted array.

direct manipulation. A user interface technique whereby the user initiates application functions by manipulating the objects, represented by icons, on the Presentation Manager (PM) or Workplace Shell desktop. The user typically initiates an action by:

1. Selecting an icon
2. Pressing and holding down a mouse button while “dragging” the icon over another object’s icon on the desktop
3. Releasing the mouse button to “drop” the icon over the target object.

Thus, this technique is also known as “drag and drop” manipulation.

direct manipulation. In Windows, a unit of data. The unit is often part of a task that is shared among users.

document component. The basic unit of data exchange in the Compound Document Framework. A document component can be either a server or a container. Document components are commonly referred to as either documents or components.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, you need hardware and supporting software that are DBCS-enabled to enter, display, and print DBCS characters.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and can be addressed as a unit. For the C Set++ for AIX compiler, a doubleword is 32 bits (4 bytes).

drag after. A target enter event that occurs in a container where its `orderedTargetEmphasis` or `mixedTargetEmphasis` attribute is set and the current view is name, text, or details.

drag item. A “proxy” for the object being manipulated.

drag over. A target enter event that occurs in a container where its `orderedTargetEmphasis` attribute is not set and the current view is icon or tree view.

drop offset. The location where the next container object that is dropped will be positioned (if the target operation’s drop style is not `IDM::dropPosition`). The position is based upon the last object that was dropped as an offset of that object relative to the drop style.

dynamic casting. An intelligent mechanism to obtain the correct pointer to a base class.

element •font

E

element. The component of an array, subrange, enumeration, or set.

element equality. A relation that determines whether two elements are equal.

element function. A function, called by a member function, that accesses the elements of a class.

embedded component. Data created by another application that is stored in a container. Unlike a linked object, an embedded component does not have its own file on disk. Instead, it is stored in the container's structured storage file.

encapsulation. The hiding of the internal representation of objects and implementation details from the client program.

enqueue. An operation that adds an element as the last element to a queue.

enumeration constant. An identifier that is defined in an enumeration and that has an associated constant integer value. You can use an enumeration constant anywhere an integer constant is allowed.

enumeration data type. A type that represents integers and a set of enumeration constants. Each enumeration constant has an associated integer value.

equality collection. (1) An abstract class with the property of element equality. (2) In general, any collection that has element equality.

equality key collection. An abstract class with the properties of element equality and key equality.

equality key sorted collection. An abstract class with the properties of element equality, key equality, and sorted elements.

equality sequence. A sequentially ordered flat collection with element equality.

equality sorted collection. An abstract class with the properties of element equality and sorted elements.

event. In Windows, a synchronization kernel object used to signal that an operation has completed. See also kernel object. Compare to critical section, mutex, semaphore, manual-reset event, and auto-reset event.

exception. (1) A user or system error detected by the system and passed to an operating system or user exception

handler. (2) For C++, any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called "throwing the exception").

exception handler. (1) A function that is invoked when an exception is detected, and that either corrects the problem and returns execution to the program, or terminates the program. (2) In C++, a catch block that catches a C++ exception when it is thrown from a function in a try block.

exception handling. A type of error handling that allows control and information to be passed to an exception handler when an exception occurs. Under the OS/2 operating system, exceptions are generated by the system and handled by user code. In C++, try, catch, and throw expressions are the constructs used to implement C++ exception handling.

external data definition. A definition appearing outside a function. The defined object is accessible to all functions that follow the definition and are located within the same source file as the definition.

eyecatcher. A recognizable sequence of bytes that determines which parameters were passed in which registers. This sequence is used for functions that have not been prototyped or have a variable number of parameters.

F

file descriptor. A small positive integer that the system uses instead of the file name to identify an open file.

file scope. A name declared outside all blocks and classes has file scope and can be used after the point of declaration in a source file.

filter. A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream.

first element. The element visited first in an iteration over a collection. Each collection has its own definition for first element. For example, the first element of a sorted set is the element with the smallest value.

flat collection. A collection that has no hierarchical structure.

folder. A directory.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

frame • inheritance

frame. (1) A complete television picture that is composed of two scanned fields, one of the even lines and one of the odd lines. In the NTSC system, a frame has 525 horizontal lines and is scanned in 1/30th of a second. (2) A border around a window.

frame extension. A control you can add if it is not available in the basic Presentation Manager frame windows.

frame number. (1) The number used to identify a frame. (2) The location of a frame on a videodisc or in a video file. On videodisc, frames are numbered sequentially from 1 to 54,000 on each side and can be accessed individually; on videotape, the numbers are assigned by way of the SMPTE time code.

frame rate. The speed at which the frames are scanned. For a videodisc player, the speed at which frames are scanned is 30 frames per second for NTSC video. For most videotape devices, the speed is 24 frames per second.

friend class. A class in which all the member functions are granted access to the private and protected members of another class. It is named in the declaration of the other class with the prefix friend.

friend function. A function that is granted access to the private and protected parts of a class. It is named in the declaration of the class with the prefix friend.

full-motion video. (1) Video playback at 30 frames per second on NTSC signals. (2) A digital video compression technique that operates in real time.

G

gain. The ability to change the audibility of the sound, such as during a fade in or fade out of music.

GDI. Graphics device interface

graphic attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string.

graphical user interface (GUI). Type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop.

graphics. A picture defined in terms of graphic primitives and graphic attributes.

Graphics device interface object. An object such as a brush, pen, or bitmap. All graphics device interface (GDI) objects are owned by the process that also owns the thread.

Compare to user object and kernel object.

GUI. Graphical user interface.

GUID. The globally unique identifier for an object. The system Registry uses GUIDs to distinguish all OLE objects available on a system. A CLSID is a type of GUID.

H

halftone. The reproduction of continuous-tone artwork, such as a photograph, by converting the image into dots of various sizes.

hash function. A function that determines which category, or bucket, to put an element in. A hash function is needed when implementing a hash table.

hash table. A data structure that divides all elements into (preferably) equal-sized categories, or buckets, to allow quick access to the elements. The hash function determines which bucket an element belongs in.

header file. A file that can contain system-defined control information or user data and generally consists of declarations.

heap. An unordered flat collection that allows duplicate elements.

height of a tree. The length of the longest path from the root to a leaf.

hit testing. The means of identifying which graphic object the mouse is pointing to.

I

implementation class. A class that implements a concrete class. Implementation classes are never used directly.

incomplete class declaration. A class declaration that does not define any members of a class. Typically, you use an incomplete class declaration as a forward declaration.

indirection. A mechanism for connecting objects by storing, in one object, a reference to another object.

inheritance. (1) A mechanism by which a derived class can use the attributes, relationships, and member functions

initializer •key sorted set

defined in more abstract classes related to it (its base classes). See also multiple inheritance. (2) An object-oriented programming technique that allows you to use existing classes as bases for creating other classes.

initializer. An expression used to initialize objects.

inlined function. A function call that the compiler replaces with the actual code for the function. You can direct the compiler to inline a function with the inline keyword.

in-place activation. The merging of the elements of user interfaces for a container and a server, such as small child windows, tool bars and menus, into the container's window space. The merger allows the user access to the server's controls from within the container, thereby providing a more document-centric approach to working. By contrast, simple activation of a server is a more application-centric approach to application interaction. Synonym for in situ editing.

input stream. A stream used to read input.

instance number. A number that the operating system uses to keep track of all of the instances of the same type of device. For example, the amplifier-mixer device name is AMPMIX plus a 2-digit instance number. If a program creates two amplifier-mixer objects, the device names could be AMPMIX01 and AMPMIX02.

integral object. A character object, an object having an enumeration type, an object having variations of the type int, or an object that is a bit field.

interactive graphics. Graphics that a user at a terminal can move or manipulate.

interactive video. The process of combining video and computer technology so that the user's actions, choices, and decisions affect the way in which the program unfolds.

interrupt. A temporary suspension of a process caused by an external event, performed in such a way that the process can be resumed.

intersection. Given collections A and B, the set of elements that is contained in both A and B.

intrinsic function. A function supplied by a program as opposed to a function supplied by the compiler.

inverted colors. Opposite colors in the light spectrum.

iteration. The process of repeatedly applying a function to a series of elements in a collection until some condition is satisfied.

iteration order. The order in which elements are accessed when iterating over a collection. In ordered collections, the element at position 1 will be accessed first, then the element at position 2, and so on. In sorted collections, the elements are accessed according to the ordering relation provided for the element type. In collections that are not ordered the elements are accessed in an arbitrary order. Each element is accessed exactly once.

iterator class. A class that provides iteration functions.

I/O Stream Library. A class library that provides the facilities to deal with many varieties of input and output.

K

kernel. The core of an operating system, usually responsible for basic I/O and process execution.

kernel object. In Windows, an object used by the system and your applications to manage numerous resources, such as processes, threads, and files.

Compare to graphics device interface object and user object.

key access. A property that allows elements to be accessed by matching keys.

key bag. An unordered flat collection that uses keys and can contain duplicate elements.

key collection. (1) An abstract class that has the property of key access. (2) In general, any collection that uses keys.

key equality. A relation that determines whether two keys are equal.

key() function. When used on a flat collection, a function that returns a reference to the key of an element.

key-type function. Any of several functions of an element type, that are used by the Collection Class Library member functions to manipulate the keys of a class.

key set. An unordered flat collection that uses keys and does not allow duplicate elements.

key sorted bag. A sorted flat collection that uses keys and allows duplicate elements.

key sorted collection. An abstract class with the properties of key equality and sorted elements.

key sorted set. A sorted flat collection that uses keys and does not allow duplicate elements.

keyword. (1) A predefined word reserved for the C or C++ language that you cannot use as an identifier. (2) A symbol that identifies a parameter.

L

last element. The element accessed last in an iteration over a collection. Each collection has its own definition for last element. For example, the last element of a sorted set is the element with the largest value.

latched. The state of a button. A button in its latched state is held in its pressed position until the user clicks on it to release (unlatch) it.

leaves. In a tree, nodes without children. Synonymous with terminals.

library. (1) A collection of functions, function calls, subroutines, or other data. (2) A set of object modules that can be specified in a link command.

linkage editor. Synonym for linker.

linked component. A component whose data is not stored directly in the compound document itself. Instead, data is stored elsewhere, and the compound document includes a name that names the other location.

linked implementation. An implementation in which each element contains a reference to the next element in the collection. Pointer chains are used to access elements in linked implementations. Linked implementations are also called linked list implementations.

linked sequence. A sequence that uses a linked implementation.

linker. A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single executable program.

locale. The definition of the subset of a user's environment that depends on language and cultural conventions.

lvalue. An expression that represents an object that can be both examined and altered.

M

manipulator. A value that can be inserted into streams or extracted from streams to affect or query the behavior of the stream.

manual-reset event. In Windows, an event used to signal several threads simultaneously that an operation has completed.

See also event.

mask. A pattern of bits or characters that controls the keeping, deleting, or testing of portions of another pattern of bits or characters.

MBCS (multibyte character set). See multibyte character set

MDI. See multiple document interface.

member. Data, functions, or types contained in classes, structures, or unions.

member function. An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and member functions of objects of its class.

message. A request from one object that the receiving object implement a method. Because data is encapsulated and not directly accessible, a message is the only way to send data from one object to another. Each message specifies the name of the receiving object, the method to be implemented, and any parameters the method needs for implementation.

method. Synonym for member function.

MIDI. Musical Instrument Digital Interface. A standard used in the music industry for interfacing digital musical instruments.

mix. (1) An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as foreground mix. Contrast with background mix. (2) The combination of audio or video sources during postproduction.

mixer. A device used to simultaneously combine and blend several inputs into one or two outputs.

mode. A collection of attributes that specifies a file's type and its access permissions.

model •nonpreemptive multitasking

model. The data portion of a document component. The model and the view comprise the two pieces of a document component. The Compound Document Framework provides an IModel base class from which you can derive your own model classes.

motion video. Video that displays real motion.

mount. (1) To place a data medium in a position to operate. (2) To make recording media accessible.

Moving Pictures Experts Group (MPEG). (1) A group that is working to establish a standard for compressing and storing motion video and animation in digital form. (2) The compression standard of video and audio data that is stored on mass media.

MPEG. Moving Pictures Experts Group.

multibyte character set (MBCS). A character set whose characters consist of more than 1 byte. Used in languages such as Japanese, Chinese, and Korean, where the 256 possible values of a single-byte character set are not sufficient to represent all possible characters.

multimedia. Computer-controlled presentations combining any of the following: text, graphics, animation, full-motion images, still video images, and sound.

multiple document interface (MDI). An interface that uses a primary window to contain related document windows. The parent window's title bar is displayed along with the child window's title bar. If the child window displays a document window, an icon that indicates the application data's file type appears in the child window's title bar.

multiple inheritance. (1) An object-oriented programming technique implemented in C++ through derivation, in which the derived class inherits members from more than one base class. (2) The structuring of inheritance relationships among classes so a derived class can use the attributes, relationships, and functions used by more than one base class.

See also inheritance and class lattice.

multitasking. (1) A mode of operation that allows concurrent performance or interleaved execution of more than one task or program. (2) A process that allows a computer or operating system to run multiple applications or tasks concurrently by dividing the processor's time between them rapidly.

See also preemptive multitasking. Contrast with nonpreemptive multitasking.

multithread. Pertaining to concurrent operation of more than one path of execution within a computer.

multithreading. A process that allows a multitasking operating system to multitask subportions (threads) of an application smoothly.

mutex. (1) In Windows, a flag that prevents threads from interacting with the 16-bit kernel when another thread is executing code there. See also nonreentrant. (2) A synchronization kernel object that synchronizes data access across multiple processes. A mutex object is either signaled or nonsignaled and is owned by a thread. See also critical section, semaphore, event, signaled, and nonsignaled.

N

n-ary tree. A tree that has an upper limit, n , imposed on the number of children allowed for a node.

National Television Standard Committee (NTSC). (1) A committee that sets the standard for color television broadcasting and video in the United States (currently in use also in Japan). (2) The standard set by the NTSC committee (the NTSC standard).

native. The rendering mechanism and format (RMF) that best represents the object and is the best one for rendering.

For example, a native of Cincinnati understands the streets in the area better than someone who has just moved there. Therefore, a Cincinnati native can get from point A to point B quicker than a newcomer. Likewise, a native RMF can get the data transferred from point A to point B more efficiently than the additional RMFs. We can use additional RMFs when we cannot use the native, or optimal, approach.

nested class. A class defined within the scope of another class.

new. (1) A C++ keyword identifying a free storage allocation operator. (2) A C++ operator used to create class objects.

new-line character. A control character that causes the print or display position to move to the first position on the next line. This control character is represented by \n in the C language.

node. In a tree structure, a point at which subordinate items of data originate.

nonpreemptive multitasking. A type of multitasking on 16-bit Windows where one application must notify the operating system that it is finished processing before the

nonreentrant • overflow

scheduler can assign another application execution time. Also called cooperative multitasking.

Contrast with preemptive multitasking.

nonreentrant. The state of 16-bit code in the kernel where two threads cannot access it at the same time without risking a system crash.

In Windows 95, processes are preempted and any thread is likely to be interrupted at any point in its execution.

See also mutex.

nonsignaled. In Windows, the state that an object is in when it is suspended, or asleep. For example, when a thread is created and running, its associated thread kernel object is nonsignaled. As soon as the thread terminates, its thread kernel object is signaled.

notification area. (1) In Visual Builder, the information or status area at the bottom of the window. (2) In Windows 95, an area to the extreme right of the taskbar. By default (on machines with a sound driver), the notification area has two objects: a loudspeaker icon and text that shows the current time.

NTFS. See NT file system.

NT file system (NTFS). A Windows NT disk drive file system that restores disk-based data after a system failure. NTFS can manipulate extremely large storage media and has file names up to 255 characters in length.

NTSC. National Television Standard Committee.

NTSC format. The specifications for color television as defined by the NTSC, which include: (a) 525 scan lines, (b) broadcast bandwidth of 4 megaHertz, (c) line frequency of 15.75 kiloHertz, (d) frame frequency of 30 frames per second, and (e) color subcarrier frequency of 3.58 megaHertz.

null character ( ). The ASCII or EBCDIC character with the hex value 00 (all bits turned off).

O

object. (1) (2) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (3) A collection of data and member functions that operate on that data, which together represent a logical entity in the system. In object-oriented programming, objects are grouped into classes that share common data definitions and member functions. Each object in the class is said to be an instance of the class. (4) In

Visual Builder, an instance of an object class consisting of attributes, a data structure, and operational member functions. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and member functions as other instances of the object class, though it has unique values assigned to its attributes. (5) In Windows, any item that is or can be linked into another Windows application, such as a sound, graphic, piece of text, or portion of a spreadsheet. An object must be from an application that supports OLE. See object linking and embedding (OLE).

object-oriented programming. A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on what data objects comprise the problem and how they are manipulated, not on how something is accomplished.

object linking and embedding (OLE). (1) An API that supports compound documents, cross-application macro control, and common object registration. OLE defines protocols for visual editing, drag-and-drop data transfers, structured storage, custom controls, and more. (2) A data sharing scheme that allows dissimilar applications to create single complex documents through a cooperative scheme. The documents can consist of material that a single application could not have created on its own.

OLE. See object linking and embedding (OLE).

operation class. A class that defines all required element and key operations required by a specific collection implementation.

operator function. An overloaded operator that is either a member of a class or that takes at least one argument that is a class type or a reference to a class type. See overloading.

optical reflective disc. An optical videodisc that is read by means of the reflection of a laser beam from the shiny surface on the disc.

ordered collection. (1) An abstract class that has the property of ordered elements. (2) In general, any collection that has its elements arranged so that there is always a first element, last element, next element, and previous element.

ordering relation. A property that determines how the elements are sorted. Ascending order is an example of an ordering relation.

overflow. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overloading • process

overloading. An object-oriented programming technique where one or more function declarations are specified for a single name in the same scope.

owner window. A window similar to a parent window, but it does not affect the behavior or appearance of the window. The owner coordinates the activity of a window.

P

pad. To fill unused positions in a field with data, usually 0's, 1's, or blanks.

parameter declaration. A description of a value that a function receives. A parameter declaration determines the storage class and the data type of the value.

parent node. A node to which one or more other nodes are subordinate.

parent window. A window that provides the child window information on how and where to draw it. The parent window also defines the relationship that the child window has with other windows in the system.

pause. To temporarily halt the medium. The halted visual should remain displayed but no audio should be played.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for pixel and picture element.

picture element. Synonym for pel.

pitch. The ability to change the key or keynote of the sound. For example, in music, the different pitches of people's voices are soprano, alto, tenor, baritone, and bass, arranged from the highest to lowest pitch.

pixel. Picture element. Synonym for pel.

pointer. A variable that holds the address of a data object or function.

pointer class. A class that implements pointers.

pointer to member. An operator used to access the address of nonstatic members of a class.

polymorphic function. A function that can be applied to objects of more than one data type. C++ implements polymorphic functions in two ways:

1. Overloaded functions (calls are resolved at compile time)
2. Virtual functions (calls are resolved at run time)

polymorphism. The technique of taking an abstract view of an object or function and using any concrete objects or arguments that are derived from this abstract view.

positioning property. The property of an element that is used to position the element in a collection. For example, the value of the key may be used as the positioning property.

precondition. A condition that a function requires to be true when it is called.

predicate function. A function that returns an IBoolean value of *true* or *false*. (IBoolean is an integer-represented Boolean type.)

preemptive multitasking. The operating system's ability to interrupt a thread at (almost) any time and assign the processor to a waiting thread. Multiple applications can thus run simultaneously, and a single application cannot control all of the system resources.

Contrast with nonpreemptive multitasking.

preparation. Any activity that the source performs before rendering the data. For example, the drag item may require that the source create a secondary thread for the source rendering to take place in. The system remains responsive to users so that they can do other tasks.

preprocessor. A phase of the compiler that examines the source program for preprocessor statements, which are then executed, resulting in the alteration of the source program.

preroll. To prepare a device to begin a playback or recording function with minimal delay.

primary thread. The first thread created when a process is initialized.

See also thread.

primitive. See graphic primitive.

primitive attribute. A specifiable characteristic of a graphic primitive. See graphic attributes.

priority queue. A queue that has a priority assigned to its elements. When accessing elements, the element with the highest priority is removed first. A priority queue has a largest-in, first-out behavior.

private. Pertaining to a class member that is accessible only to member functions and friends of that class.

process. (1) A collection of code, data, and other system resources, including at least one thread of execution, that

performs a data processing task. (2) A running application, its address space, and its resources. (3) An instance of a running program. A Win32 process owns a 4-GB address space containing the code and data for an application's .exe file; it does not execute anything. It also owns certain resources, such as files, dynamic memory allocations, and threads. (4) A program running under OS/2, along with the resources associated with it (memory, threads, file system resources, and so on).

profiling. The process of generating a statistical analysis of a program that shows processor time and the percentage of program execution time used by each procedure in the program.

program. (1) One or more files containing a set of instructions conforming to a particular programming language syntax. (2) A self-contained, executable module. Multiple copies of the same program can be run in different processes.

Program Files. A folder, which Windows 95 Setup places off the root of the drive on which Windows 95 is installed. It makes a subfolder called Accessories and places files such as WordPad and Paint there. This is the recommended default location for application programs.

project. A container that groups related objects (tasks) into a primary window. When the user opens the object, the object has its own primary window.

property function. A function that is used to determine whether the element it is applied to has a given property or characteristic. A property function can be used, for example, to remove all elements with a given property.

protected. Pertaining to a class member that is only accessible to member functions and friends of that class, or to member functions and friends of classes derived from that class.

prototype. A function declaration or definition that includes both the return type of the function and the types of its arguments.

public. Pertaining to a class member that is accessible to all functions.

pure virtual function. A virtual function that has a function initializer of the form `= 0`;

Q

queue. A sequence with restricted access in which elements can only be added at the back end (or bottom) and removed from the front end (or top). A queue is characterized by first-in, first-out behavior and chronological order.

R

reference class. A class that links a concrete class to an abstract class. Reference classes make polymorphism possible with the Collection Classes.

relation. An unordered flat collection class that uses keys, allows for duplicate elements, and has element equality.

renderer. An object that renders data using a particular mechanism, such as using files or shared memory. It contains definitions of supported rendering mechanisms and formats and types. Renderers are maintained positionally (1-based).

rendering. The transfer or re-creation of the dragged object from the source window to the target window.

rendering format. Identifies the actual format of the data being rendered in a direct manipulation operation.

rendering mechanism. Identifies the actual format of the data being rendered in a direct manipulation operation.

resource file. A file that contains data used by an application, such as text strings and icons.

returned element. An element returned by a function as the return value.

RGB. Red, green, blue. A method of processing color images according to their red, green, and blue color content.

RMFs. Rendering mechanisms and formats.

root. A node that has no parent. All other nodes of a tree are descendants of the root.

samples-per-second •step forward

S

samples-per-second. The number of times per second that the audio card records data from the audio input. For example, 44 kiloHertz is CD quality; 22 kiloHertz is FM music quality; and 11 kiloHertz is voice quality.

SBCS (Single-Byte Character Set). See single-byte character set

scan. To search backward and forward at high speed on a CD audio device. Scanning is analogous to fast forwarding.

scope. That part of a source program in which an object is defined and recognized.

scope operator (::). An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.

scroll increment. The number by which the current value of the circular slider is incremented or decremented when a user presses one of the circular slider control buttons.

semaphore. A synchronization kernel object used for resource-counting. A semaphore offers a thread the ability to query the number of resources available. If one or more resources are available, the count of available resources is decremented.

See also critical section, mutex, and event.

sequence. A sequentially ordered flat collection.

sequential collection. An abstract class with the property of sequentially ordered elements.

server. An application, or document component, that supplies an object, as opposed to a container, which contains objects. For example, a drawing program that provides a picture that can be placed inside a word-processing document is referred to as a server.

siblings. All the children of a node are said to be siblings of one another.

signaled. A state in which an object has been reactivated after the threads have been put to sleep. For example, if a thread in a parent process needs to wait for the child process to terminate, the parent's thread puts itself to sleep until the kernel object identifying the child process becomes signaled.

single-byte character set (SBCS). A set of characters in which each character is represented by a 1-byte code.

SMPTE time code. A frame-numbering system developed by SMPTE that assigns a number to each frame of video. The 8-digit code is in the form HH:MM:SS:FF (hours, minutes, seconds, frame number). The numbers track elapsed hours, minutes, seconds, and frames from any chosen point.

sorted bag. A sorted flat collection that allows duplicate elements.

sorted collection. (1) An abstract class with the property of sorted elements. (2) In general, any collection with sorted elements.

sorted map. A sorted flat collection with key and element equality.

sorted relation. A sorted flat collection that uses keys, has element equality, and allows duplicate elements.

sorted set. A sorted flat collection with element equality.

sound track. Synonymous with audio track.

sprite. A small graphic that can be moved independently around the screen, producing animated effects.

stack. A data structure in which new elements are added to and removed from the top of the structure. A stack is characterized by Last-In-First-Out (LIFO) behavior.

standard error. An output stream usually intended to be used for diagnostic messages.

standard input. An input stream usually intended to be used for primary data input. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. An output stream usually intended to be used for primary data output. When programs are run interactively, standard output usually goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

step backward. In multimedia applications, to move the medium backward one frame or segment at a time.

step forward. In multimedia applications, to move the medium forward one frame or segment at a time.

step frame •transparent color

step frame. A function of devices such as digital video and videodisc players that enables a user to move frame-by-frame in either direction.

storage. Like a directory in a conventional file system, storage manages other storages and streams but holds no data itself. Storage objects works with stream objects to provide persistent storage. A stream acts like a file in that it can hold information but not other storage elements.

stream. (1) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. (2) A file access object that allows access to an ordered sequence of characters, as described by the ISO C standard. A stream provides the additional services of user-selectable buffering and formatted input and output. (3) Like a single disk file in a conventional file system, a stream is kept in a storage object, which is like a directory in a conventional file system. A stream is named using a text string, and it can have any internal structure that you define.

stream buffer. A stream buffer is a buffer between the ultimate consumer, ultimate producer, and the I/O Stream Library functions that format data. It is implemented in the I/O Stream Library by the streambuf class and the classes derived from streambuf.

string. A contiguous sequence of characters.

structure. A construct that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

structured storage. A standardized means of accessing units of information using storages and streams. Conceptually, structured storage is like a file system within a file that allows diverse collections of data to be stored in a single file.

subclass. See derived class.

subscript. One or more expressions, each enclosed in brackets, that follow an array name. A subscript refers to an element in an array.

subtree. A tree structure created by arbitrarily denoting a node to be the root node in a tree. A subtree is always part of a whole tree.

superclass. See base class and abstract class.

superset. Given two sets A and B, A is a superset of B if and only if all elements of B are also elements of A. That is, A is a superset of B if B is a subset of A.

T

tabular implementation. An implementation that stores the location of elements in tables. Elements in a tabular implementation are accessed by using indices to arrays.

tabular sequence. A sequence that uses a tabular implementation.

taskbar. In Windows 95, a bar at the bottom (default position) of the desktop that shows all open applications and active windows. Clicking on any task button brings the corresponding session to the foreground.

template. A family of classes or functions where the code remains invariant but operates with variable types.

terminals. Synonym for *leaves*.

this. A C++ keyword that identifies a special type of pointer in a member function, one that references the class object with which the member function was invoked.

this collection. The collection to which a function is applied.

thread. (1) The atomic unit or path of execution within a process, a piece of executing code. (2) In Windows, each thread is located in its own stack from the owning process' 4-GB address space, and each one has its own set of processor registers, called the thread's context. See also primary thread and zero page thread.

thread synchronization. The ability to synchronize the activities of various threads. A thread synchronizes itself with another thread by putting itself to sleep. Before doing so, the thread notifies the operating system as to what event has to occur in order for the thread to resume execution.

throw expression. An argument to the C++ exception being thrown.

time code. See SMPTE time code.

tool bar. The area under the title bar that displays the tools available.

transparency. Refers to when a selected color on a graphics screen is made transparent to allow the video behind it to become visible.

transparent color. (1) A clear color used to indicate the part of the bitmap that is not drawn for the bitmap. The area under the bitmap is not overpainted for areas of the bitmap that are set to the transparent color. (2) Video information is

trap •video graphics adapter (VGA)

considered as being present on the video plane that is maintained behind the graphics plane. When an area on the graphics plane is painted with a transparent color, the video information in the video plane is made visible.

trap. An unprogrammed conditional jump to a specified address that is automatically activated by hardware. A recording is made of the location from which the jump occurred.

treble. (1) The upper half of the whole vocal or instrumental tonal range. (2) The higher portion of the audio frequency range in sound recording.

tree. A hierarchical collection of nodes that can have an arbitrary number of references to other nodes. A unique path connects every two nodes.

tree control. A type of control that shows the hierarchical relationships among a set of objects by indenting them as in an outline. The user can expand or collapse the various branches (levels) of the tree (outline).

true and additional. The most accurate or most descriptive (primary) type of an object (true) and the other or secondary types (additional). For example, if the object is a text file, its true type is text; if the file was a C source code file, its true type is C code.

try block. A block in which a known C++ exception is passed to a handler.

typed implementation class. A class that implements a concrete class and provides an interface that is specific to a given element type. This interface allows the compiler to verify that, for example, integers cannot be added to a set of strings.

typeless implementation class. A class that implements a concrete class and provides an interface that is not specific to a given element type.

U

ultimate consumer. The target of data in an I/O operation. An ultimate consumer can be a file, a device, or an array of bytes in memory.

ultimate producer. The source of data in an I/O operation. An ultimate producer can be a file, a device, or an array of bytes in memory.

unbounded collection. A collection that has no upper limit on the number of elements it can contain.

undefined cursor. A cursor that may or may not be valid, and that may or may not refer to a different element of the collection from the element it referred to before the function call that resulted in its becoming undefined. An undefined cursor may refer to no element of the collection, and still be a valid cursor.

underflow. (1) A condition that occurs when the result of an operation is less than the smallest possible nonzero number. (2) Synonym for arithmetic underflow, monadic operation.

union. (1) Structures that can contain different types of objects at different times. Only one of the member objects can be stored in a union at any time. (2) Given the sets A and B, all elements of A, B, or both A and B.

unique collection. A collection in which the value of an element only occurs once; that is, there are no duplicate elements.

unload. To eject the medium from the device.

unordered collection. A collection that has no order to its elements.

user object. An object, such as an icon, a window, a menu, or an accelerator table. In Windows, most user objects are owned by the thread that created them (icons, cursors, and windows classes are owned by a process).

V

VCR. Videocassette recorder.

VGA. Video graphics adapter.

view. The user interface controls and associated handlers for a document component. The view and the model comprise the two pieces of a document component. The Compound Document Framework provides an IView base class from which you can derive your own view classes.

video. Pertaining to the portion of recorded information that can be seen.

video attributes. The standard video attributes are: brightness, contrast, freeze, hue, saturation, and sharpness.

video graphics adapter (VGA). A graphics controller for color displays. The pixel resolution of the video graphics adapter is 4:4.

videocassette recorder (VCR) •(:) (double colon)

videocassette recorder (VCR). A device for recording or playing back videocassettes.

videodisc. A disc on which programs have been recorded for playback on a computer or a television set; a recording on a videodisc. The most common format in the United States and Japan is an NTSC signal recorded on the optical reflective format.

videodisc player. A device that provides video playback for prerecorded videodiscs.

virtual function. A function of a class that is declared with the keyword `virtual`. The implementation that is executed when you make a call to a virtual function depends on the type of the object for which it is called. This is determined at run time.

volatile. An attribute of a data object that indicates the object is changeable beyond the control or detection of the compiler. Any expression referring to a volatile object is evaluated immediately, for example, assignments.

volume. The intensity of sound. A volume of 0 is minimum volume. A volume of 100 is maximum volume.

W

white space. Space characters, tab characters, form feed characters, and new-line characters.

wide character. A character whose range of values can represent distinct codes for all members of the largest extended character set specified among the supporting locales.

Win32. The name of an application programming interface (API).

See also Win32 API.

Win32 API. (1) A set of Win32 functions that can be called from source code. (2) A 32-bit version of the 16-bit Windows 3.1 API (native to Windows NT).

See also Win32.

Win32s. A platform that the Win32 API is implemented on. (The *s* stands for subset.) It consists of a virtual-device driver and dynamic link libraries (DLLs) that add the Win32 API to the 16-bit Windows 3.x system. It includes structured exception handling and limited implementations of memory-mapped files.

See also Win32 API and Windows 95.

Windows NT. A platform that the Win32 API is implemented on. It is a portable, high-end operating system, which can run several different types of applications simultaneously. It is the only Win32 platform for machine architectures based on processors other than the x86, and it supports multiple processors.

See also Win32 API.

Windows 95. (1) A 32-bit operating system that allows you to run 32-bit application. Windows 95 is a multitasking, multithreaded operating system that can control multiple programs at once. Each program can have multiple concurrent threads or independently executing subcomponents. (2) A platform that the Win32 API is implemented on. It supports image color matching, modems, and other services. It partially supports asynchronous file I/O, debugging, registry, security, and event-logging functions.

workspace. A container object that provides for the association and management of task-related windows within a parent window.

Z

zero page thread. A thread with a priority level of 0 that zeros out any free pages in the system when there are no other threads that need to perform work in the system. No other threads can have a priority level of 0.

See also thread and primary thread.

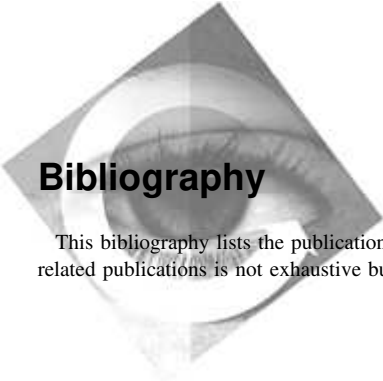
Numerics

24-bit color. A digital standard that uses 24 bits of information to describe each color pixel, providing up to 16.7 million colors in one image (the highest digital standard currently available).

8-bit color. A digital standard that uses 8 bits of information to describe each color pixel, providing up to 256 colors in one image (the standard for VGA displays).

Special Characters

(::) (double colon). Scope operator. An operator that defines the scope for the argument on the right. If the left argument is blank, the scope is global; if the left argument is a class name, the scope is within that class. Also called a scope resolution operator.



Bibliography

This bibliography lists the publications that make up the IBM VisualAge for C++ library and related publications. The list of related publications is not exhaustive but should be adequate for most VisualAge for C++ users.

The IBM VisualAge for C++ Library

The following books are part of the IBM VisualAge for C++ library.

- *Installation Guide & Product Overview*, S33H-5030
- *User's Guide*, S33H-5031
- *Programming Guide*, S33H-5032
- *Visual Builder User's Guide*, S33H-5034
- *Visual Builder Parts Reference*, S33H-5035
- *Building VisualAge for C++ Parts for Fun and Profit*, S33H-5036
- *Open Class Library User's Guide*, S33H-5033
- *Open Class Library Reference*, S33H-5039
- *Language Reference*, S33H-5037-00
- *C Library Reference*, S33H-5038
- *SOM Programming Guide*, S33H-5044
- *SOM Programming Reference*, SOM Programming Reference

C and C++ Related Publications

- *Portability Guide for IBM C*, SC09-1405
- *American National Standard for Information Systems / International Standards Organization — Programming Language C (ANSI/ISO 9899-1990[1992])*

Non-IBM Publications

Many books have been written about the C++ language and related programming topics. The authors use varying approaches and emphasis. The following is a sample of some

non-IBM C++ publications that are generally available. This sample is not an exhaustive list. IBM does not specifically recommend any of these books, and other C++ books may be available in your locality.

- *The Annotated C++ Reference Manual* by Margaret A. Ellis and Bjarne Stroustrup, Addison-Wesley Publishing Company.
- *C++ Primer* by Stanley B. Lippman, Addison-Wesley Publishing Company.
- *Object-Oriented Design with Applications* by Grady Booch, Benjamin/Cummings.
- *Object-Oriented Programming Using SOM and DSOM* by Christina Lau, Van Nostrand Reinhold..
- *OS/2 C++ Class Library: Power GUI Programming with C Set ++* by Kevin Leong, William Law, Robert Love, Hiroshi Tsuji, and Bruce Olson, John Wiley and Sons Publishing Company.

Suggested Reading for Collection

Classes: These books contain explanations of data structures that may help you understand the data structures in the Collection Classes:

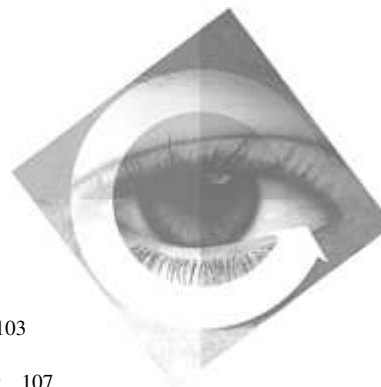
- *Data Structures and Algorithms* by Aho, Hopcroft, and Ullman, Addison-Wesley Publishing Company.
- *The Art of Computer Programming, Vol. 3: Sorting and Searching*, D.E. Knuth, Addison-Wesley Publishing Company.
- *C++ Components and Algorithms* by Scott Robert Ladd, M&T Publishing Inc.
- *A Systematic Catalogue of Reusable Abstract Data Types* by Juergen Uhl and Hans Albrecht Schmit, Springer Verlag.

Index

Special Characters

~ColumnCursor
 IContainerControl::ColumnCursor 238
~CompareFn
 IContainerControl::CompareFn 240
~Cursor
 INotebook::Cursor 526
~FilterFn
 IContainerControl::FilterFn 243
~FrameCursor
 IToolBar::FrameCursor 627
~IAnimatedButton
 IAnimatedButton 11
~IBitmapControl
 IBitmapControl 20
~ICanvas
 ICanvas 28
~ICircularSlider
 ICircularSlider 43
~ICircularSliderNotifyHandler
 ICircularSliderNotifyHandler 52
~ICnrAllocator
 ICnrAllocator 55
~ICnrBeginEditEvent
 ICnrBeginEditEvent 58
~ICnrControlList
 ICnrControlList 60
~ICnrDrawBackgroundEvent
 ICnrDrawBackgroundEvent 64
~ICnrDrawHandler
 ICnrDrawHandler 68
~ICnrDrawItemEvent
 ICnrDrawItemEvent 73
~ICnrEditEvent
 ICnrEditEvent 77
~ICnrEditHandler
 ICnrEditHandler 82
~ICnrEmphasisEvent
 ICnrEmphasisEvent 88
~ICnrEndEditEvent
 ICnrEndEditEvent 90
~ICnrEnterEvent
 ICnrEnterEvent 94
~ICnrEvent
 ICnrEvent 96

~ICnrHandler
 ICnrHandler 99
~ICnrHelpEvent
 ICnrHelpEvent 103
~ICnrMenuHandler
 ICnrMenuHandler 107
~ICnrObjectSet
 ICnrObjectSet 112
~ICnrQueryDeltaEvent
 ICnrQueryDeltaEvent 114
~ICnrReallocStringEvent
 ICnrReallocStringEvent 117
~ICnrScrollEvent
 ICnrScrollEvent 121
~ICollectionViewComboBox
 ICollectionViewComboBox 129
~ICollectionViewListBox
 ICollectionViewListBox 145
~IContainerColumn
 IContainerColumn 156
~IContainerControl
 IContainerControl 191
~IContainerControlNotifyHandler
 IContainerControlNotifyHandler 255
~IContainerObject
 IContainerObject 260
~ICustomButton
 ICustomButton 272
~ICustomButtonDrawEvent
 ICustomButtonDrawEvent 282
~ICustomButtonDrawHandler
 ICustomButtonDrawHandler 285
~IDDEAcknowledgeEvent
 IDDEAcknowledgeEvent 290
~IDDEAcknowledgeExecuteEvent
 IDDEAcknowledgeExecuteEvent 294
~IDDEAcknowledgePokeEvent
 IDDEAcknowledgePokeEvent 297
~IDDEActiveServer
 IDDEActiveServer 299
~IDDEActiveServerSet
 IDDEActiveServerSet 302
~IDDEBeginEvent
 IDDEBeginEvent 303
~IDDEClientAcknowledgeEvent
 IDDEClientAcknowledgeEvent 307



- ~IDDEClientConversation
 - IDDEClientConversation 311
- ~IDDEClientEndEvent
 - IDDEClientEndEvent 324
- ~IDDEClientHotLinkEvent
 - IDDEClientHotLinkEvent 327
- ~IDDEClientHotLinkSet
 - IDDEClientHotLinkSet 330
- ~IDDEDataEvent
 - IDDEDataEvent 332
- ~IDDEEndEvent
 - IDDEEndEvent 334
- ~IDDEEvent
 - IDDEEvent 338
- ~IDDEExecuteEvent
 - IDDEExecuteEvent 341
- ~IDDEPokeEvent
 - IDDEPokeEvent 344
- ~IDDERequestDataEvent
 - IDDERequestDataEvent 346
- ~IDDEServerAcknowledgeEvent
 - IDDEServerAcknowledgeEvent 349
- ~IDDEServerHotLinkEvent
 - IDDEServerHotLinkEvent 352
- ~IDDESetAcknowledgeInfoEvent
 - IDDESetAcknowledgeInfoEvent 356
- ~IDDETopicServer
 - IDDETopicServer 359
- ~IDrawingCanvas
 - IDrawingCanvas 371
- ~IDialog
 - IDialog 381
- ~IDialogEvent
 - IDialogEvent 398
- ~IDialogHandler
 - IDialogHandler 401
- ~IFlyOverHelpHandler
 - IFlyOverHelpHandler 408
- ~IFlyText
 - IFlyText 416
- ~IFontDialog
 - IFontDialog 422
- ~IFontDialogHandler
 - IFontDialogHandler 436
- ~IGraphicPushButton
 - IGraphicPushButton 442
- ~IIconControl
 - IIconControl 453
- ~IInfoArea
 - IInfoArea 461
- ~IMultiCellCanvas
 - IMultiCellCanvas 477
- ~INotebook
 - INotebook 497
- ~INotebookDrawItemEvent
 - INotebookDrawItemEvent 540
- ~INotebookNotifyHandler
 - INotebookNotifyHandler 543
- ~IPageEvent
 - IPageEvent 547
- ~IPageHandler
 - IPageHandler 552
- ~IPageHelpEvent
 - IPageHelpEvent 557
- ~IPageRemoveEvent
 - IPageRemoveEvent 560
- ~IPageSelectEvent
 - IPageSelectEvent 563
- ~ISetCanvas
 - ISetCanvas 567
- ~ISplitCanvas
 - ISplitCanvas 588
- ~IStringGenerator
 - IStringGenerator 599
- ~IStringGeneratorFn
 - IStringGeneratorFn 602
- ~IStringGeneratorMemberFn
 - IStringGeneratorMemberFn 605
- ~IStringGeneratorRefMemberFn
 - IStringGeneratorRefMemberFn 608
- ~Iterator
 - IContainerControl::Iterator 244
- ~IToolBar
 - IToolBar 611
- ~IToolBarButton
 - IToolBarButton 635
- ~IToolBarContainer
 - IToolBarContainer 649
- ~IToolBarFrameWindow
 - IToolBarFrameWindow 656
- ~IViewPort
 - IViewPort 663
- ~ObjectCursor
 - IContainerControl::ObjectCursor 247
- ~PageSettings
 - INotebook::PageSettings 532
- ~Settings
 - IFileDialog::Settings 391
 - IFontDialog::Settings 431

- ~TextCursor
 - ICContainerControl::TextCursor 252
- ~WindowCursor
 - IToolBar::WindowCursor 630

A

- acceptConversation
 - IDDETopicServer 365
- acknowledged
 - IDDEClientConversation 319
 - IDDETopicServer 365
- addAsFirst
 - IToolBar 617
- addAsLast
 - IToolBar 617
- addAsNext
 - IToolBar 617
- addAsPrevious
 - IToolBar 617
- addColumn
 - ICContainerControl 192
- addDrive
 - IFileDialog::Settings 393
- addFileType
 - IFileDialog::Settings 393
- addFirstPage
 - INotebook 499
- addId
 - ICContainerControl 229
- addLastPage
 - INotebook 499
- addObject
 - ICContainerControl 178
- addObjectAfter
 - ICContainerControl 178
- addObjects
 - ICContainerControl 179
- addObjectsAfter
 - ICContainerControl 179
- addPageAfter
 - INotebook 500
- addPageBefore
 - INotebook 501
- addSourceEmphasis
 - ICnrMenuHandler 108
- addToCell
 - IMultiCellCanvas 474
- Advanced Control, Dialog, and Handler Classes xviii
 - overview xviii
- alignBottom
 - ICContainerColumn 171
- alignCentered
 - ICContainerColumn 171
- alignLeft
 - ICContainerColumn 171
- alignment
 - ISetCanvas 569
- alignRight
 - ICContainerColumn 171
- alignTitleCentered
 - ICContainerControl 226
- alignTitleLeft
 - ICContainerControl 226
- alignTitleRight
 - ICContainerControl 226
- alignTop
 - ICContainerColumn 171
- alignVerticallyCentered
 - ICContainerColumn 171
- allObjectsDo
 - ICContainerControl 199
- allowsDragDelete
 - IToolBarButton 636
- allowsDragDrop
 - IToolBar 612
- allTabsVisible
 - INotebook 518
- alwaysHorizontalScrollBar
 - IViewPort 669
- alwaysVerticalScrollBar
 - IViewPort 670
- amount
 - ICnrScrollEvent 121
- IAnimatedButton 7
 - IAnimatedButton::Style 16
- animateWhenLatched
 - IAnimatedButton 14
- animationRate
 - IAnimatedButton 8
- application
 - IDDEActiveServer 300
 - IDDEBeginEvent 304
 - IDDEClientConversation 312
 - IDDEClientEndEvent 324
 - IDDETopicServer 361
- applicationSpecificData
 - IDDEAcknowledgeEvent 290
- applyButton
 - IFileDialog 386

- applyButton (*continued*)
 - IFontDialog 427
- applyTo
 - IContainerControl::Iterator 245
- areAllTabsVisible
 - INotebook 511
- areChildrenReversed
 - ICanvas 33
- areDetailsViewTitlesVisible
 - IContainerControl 180
- armRange
 - ICircularSlider 39
- arrangeIconView
 - IContainerControl 188
- asNeededHorizontalScrollBar
 - IViewport 670
- asNeededVerticalScrollBar
 - IViewport 670
- atBottomDelta
 - ICnrQueryDeltaEvent 114
- atFirstObject
 - ICnrQueryDeltaEvent 114
- atLastObject
 - ICnrQueryDeltaEvent 114
- atTopDelta
 - ICnrQueryDeltaEvent 114
- attributes
 - IContainerControl 223
- autoLatch
 - ICustomButton 278
- autoPageSize
 - INotebook::PageSettings 535
- autoPosition
 - IContainerControl 231

B

- backgroundColor
 - ICanvas 27
 - IContainerControl 206
 - ICustomButton 271
 - INotebook 493
- backPagesBottomLeft
 - INotebook 518
- backPagesBottomRight
 - INotebook 518
- backPagesTopLeft
 - INotebook 518
- backPagesTopRight
 - INotebook 518

- baseRecord
 - IContainerObject 267
- baseRecordSize
 - IContainerControl 224
- begin
 - IDDEClientConversation 313
- beginConversation
 - IDDETopicServer 360
- beginEdit
 - ICnrEditHandler 83
- beginHotLink
 - IDDEClientConversation 314
 - IDDETopicServer 365
- binding
 - INotebook 492
- bitmap
 - IAnimatedButton 9
 - IBitmapControl 17
 - IGraphicPushButton 442
 - IToolBarButton 633
- bitmapAndTextVisible
 - IToolBarButton 643
- IBitmapControl 17
- IBitmapControl::Style 25
- bitmapCount
 - IAnimatedButton 9
- bitmapOnly
 - IFontDialog 427
- bitmapSize
 - IToolBarButton 633
- bitmapVisible
 - IToolBarButton 643
- border
 - ISetCanvas 575
- bottomAlign
 - ISetCanvas 575
- buffer
 - IDDEEvent 339
- buttonBitmapAndTextVisible
 - IToolBar 623
- buttonBitmapVisible
 - IToolBar 623
- buttonPressedId
 - IFileDialog 382
 - IFontDialog 423
- buttons
 - ICircularSlider 46
- buttonState
 - ICustomButtonDrawEvent 282

- buttonTextVisible
 - IToolBar 624
- buttonView
 - IToolBar 616

C

- calcMinimumSize
 - IAnimatedButton 13
 - IBitmapControl 22
 - ICanvas 34
 - ICircularSlider 45
 - IContainerControl 224
 - ICustomButton 277
 - IDrawingCanvas 374
 - IGraphicPushButton 446
 - IIconControl 456
 - IInfoArea 467
 - INotebook 515
 - IToolBar 622
 - IToolBarButton 642
 - IViewport 667
- cancel
 - IFileDialog 386
 - IFontDialog 426
- ICanvas 26
- ICanvas::Style 37
- centerAlign
 - ISetCanvas 575
- centerVerticalAlign
 - ISetCanvas 576
- changed
 - ICnrEmphasisEvent 88
- circularArm
 - ICircularSlider 46
- ICircularSlider 38
- ICircularSlider::Style 50
- ICircularSliderNotifyHandler 51
- classDefaultAttribute
 - IContainerControl 226
- classDefaultDataStyle
 - IContainerColumn 168
- classDefaultHeadingStyle
 - IContainerColumn 170
- classDefaultStyle
 - IAnimatedButton 14
 - IBitmapControl 23
 - ICanvas 35
 - ICircularSlider 47
 - IContainerControl 231

- classDefaultStyle (*continued*)
 - ICustomButton 278
 - IDrawingCanvas 374
 - IFileDialog 386
 - IFontDialog 427
 - IGraphicPushButton 447
 - IIconControl 457
 - IMultiCellCanvas 484
 - INotebook 518
 - ISetCanvas 576
 - ISplitCanvas 593
 - IToolBar 624
 - IToolBarButton 643
 - IToolBarContainer 652
 - IViewport 670

Classes

- IAnimatedButton 7
- IAnimatedButton::Style 16
- IBitmapControl 17
- IBitmapControl::Style 25
- ICanvas 26
- ICanvas::Style 37
- ICircularSlider 38
- ICircularSlider::Style 50
- ICircularSliderNotifyHandler 51
- ICnrAllocator 54
- ICnrBeginEditEvent 57
- ICnrControlList 60
- ICnrDate 62
- ICnrDrawBackgroundEvent 63
- ICnrDrawHandler 66
- ICnrDrawItemEvent 72
- ICnrEditEvent 76
- ICnrEditHandler 81
- ICnrEmphasisEvent 87
- ICnrEndEditEvent 90
- ICnrEnterEvent 93
- ICnrEvent 96
- ICnrHandler 98
- ICnrHelpEvent 103
- ICnrMenuHandler 106
- ICnrObjectSet 111
- ICnrQueryDeltaEvent 113
- ICnrReallocStringEvent 116
- ICnrScrollEvent 120
- ICnrTime 123
- ICollectionViewComboBox 124
- ICollectionViewConstants 137
- ICollectionViewListBox 140
- IContainerColumn 153

Classes (*continued*)

IContainerControl 174
 IContainerControl::Attribute 235
 IContainerControl::ColumnCursor 236
 IContainerControl::CompareFn 240
 IContainerControl::FilterFn 242
 IContainerControl::Iterator 244
 IContainerControl::ObjectCursor 246
 IContainerControl::Style 250
 IContainerControl::TextCursor 251
 IContainerControlNotifyHandler 255
 IContainerObject 258
 ICustomButton 270
 ICustomButton::Style 280
 ICustomButtonDrawEvent 281
 ICustomButtonDrawHandler 285
 IDDEAcknowledgeEvent 289
 IDDEAcknowledgeExecuteEvent 293
 IDDEAcknowledgePokeEvent 296
 IDDEActiveServer 299
 IDDEActiveServerSet 301
 IDDEBeginEvent 303
 IDDEClientAcknowledgeEvent 306
 IDDEClientConversation 309
 IDDEClientEndEvent 323
 IDDEClientHotLinkEvent 326
 IDDEClientHotLinkSet 329
 IDDEDataEvent 331
 IDDEEndEvent 334
 IDDEEvent 337
 IDDEExecuteEvent 340
 IDDEPokeEvent 343
 IDDERequestDataEvent 346
 IDDEServerAcknowledgeEvent 349
 IDDEServerHotLinkEvent 352
 IDDESetAcknowledgeInfoEvent 355
 IDDETopicServer 358
 IDrawingCanvas 369
 IDrawingCanvas::Style 376
 IFileDialog 378
 IFileDialog::Settings 390
 IFileDialog::Style 396
 IFileDialogEvent 398
 IFileDialogHandler 400
 IFlyOverHelpHandler 406
 IFlyText 415
 IFontDialog 419
 IFontDialog::Settings 430
 IFontDialog::Style 434
 IFontDialogHandler 435

Classes (*continued*)

IGraphicPushButton 439
 IGraphicPushButton::Style 450
 IIconControl 451
 IIconControl::Style 459
 IInfoArea 460
 IMultiCellCanvas 470
 IMultiCellCanvas::Style 487
 INotebook 489
 INotebook::Cursor 525
 INotebook::PageSettings 529
 INotebook::PageSettings::Attribute 537
 INotebook::Style 539
 INotebookDrawItemEvent 540
 INotebookNotifyHandler 543
 IPageEvent 546
 IPageHandle 549
 IPageHandler 551
 IPageHelpEvent 556
 IPageRemoveEvent 559
 IPageSelectEvent 562
 ISetCanvas 565
 ISetCanvas::Style 581
 ISplitCanvas 583
 ISplitCanvas::Style 595
 IStringGenerator 597
 IStringGeneratorFn 601
 IStringGeneratorMemberFn 604
 IStringGeneratorRefMemberFn 607
 IToolBar 610
 IToolBar::FrameCursor 627
 IToolBar::Style 629
 IToolBar::WindowCursor 630
 IToolBarButton 632
 IToolBarButton::Style 647
 IToolBarContainer 648
 IToolBarContainer::Style 654
 IToolBarFrameWindow 655
 IViewport 660
 IViewport::Style 673
 clearDefaultTransparentColor
 IToolBarButton 639
 clientHandle
 IDDEClientConversation 312
 closeEdit
 IContainerControl 196
 ICnrAllocator 54
 ICnrBeginEditEvent 57
 ICnrControlList 60

- ICnrDate 62
- ICnrDrawBackgroundEvent 63
- ICnrDrawHandler 66
- ICnrDrawItemEvent 72
- ICnrEditEvent 76
- ICnrEditHandler 81
- ICnrEmphasisEvent 87
- ICnrEndEditEvent 90
- ICnrEnterEvent 93
- ICnrEvent 96
- ICnrHandler 98
- ICnrHelpEvent 103
- ICnrMenuHandler 106
- ICnrObjectSet 111
- ICnrQueryDeltaEvent 113
- ICnrReallocStringEvent 116
- ICnrScrollEvent 120
- ICnrTime 123
- collapse
 - IContainerControl 215
 - IToolBar 613
- collapseTree
 - IContainerControl 220
- collectionReplaced
 - ICollectionViewComboBox 126
 - ICollectionViewListBox 142
- ICollectionViewComboBox 124
- ICollectionViewConstants 137
- ICollectionViewListBox 140
- column
 - ICnrDrawItemEvent 73
 - ICnrEditEvent 77
 - ICnrHelpEvent 104
 - IContainerControl 224
- columnAt
 - IContainerControl 192
- columnCount
 - IContainerControl 192
- ColumnCursor
 - IContainerControl::ColumnCursor 237
- columnInfo
 - IContainerColumn 167
- columnUnderPoint
 - IContainerControl 204
- columnWidth
 - IMultiCellCanvas 479
- comboDelete
 - ICollectionViewConstants 137
- comboDeleteAll
 - ICollectionViewConstants 138
- comboInsert
 - ICollectionViewConstants 138
- commands
 - IDDEAcknowledgeExecuteEvent 294
 - IDDEExecuteEvent 341
- CompareFn
 - IContainerControl::CompareFn 240
- Compound Document Framework Classes xviii
 - overview xviii
- container
 - ICnrDrawBackgroundEvent 64
 - ICnrDrawItemEvent 73
 - ICnrEditEvent 77
 - IContainerColumn 167
- containerAttributes
 - IContainerControl 224
- IContainerColumn 153
- IContainerControl 174
 - IContainerControl::Attribute 235
 - IContainerControl::ColumnCursor 236
 - IContainerControl::CompareFn 240
 - IContainerControl::FilterFn 242
 - IContainerControl::Iterator 244
 - IContainerControl::ObjectCursor 246
 - IContainerControl::Style 250
 - IContainerControl::TextCursor 251
 - IContainerControlNotifyHandler 255
- containerFromHandle
 - IContainerControl 199
- containerId
 - ICnrEvent 97
- containerKey
 - IContainerControl 224
- containerList
 - IContainerControl 224
- IContainerObject 258
- containsObject
 - IContainerControl 212
- conversationCount
 - IDDETopicServer 361
- conversationEnded
 - IDDEClientConversation 319
 - IDDETopicServer 366
- convertToGUIStyle
 - IAnimatedButton 11
 - IBitmapControl 20
 - ICanvas 30
 - ICircularSlider 44
 - IContainerControl 219
 - ICustomButton 274

- convertToGUIStyle (*continued*)
 - IDrawingCanvas 372
 - IFileDialog 383
 - IFontDialog 424
 - IGraphicPushButton 444
 - IIconControl 454
 - IMultiCellCanvas 482
 - INotebook 510
 - ISetCanvas 571
 - ISplitCanvas 590
 - IToolBar 615
 - IToolBarButton 638
 - IToolBarContainer 650
 - IToolBarFrameWindow 656
 - IViewPort 664
- convertToWorkspace
 - IContainerControl 204
- copyObjectTo
 - IContainerControl 202
- cpp files, description xix
- CPP.NDX xx
- CPP*.INF xx
- CPPBRS.NDX xx
- CPPWO*.DEF xx
- CPPWO*.DLL xx
- CPPWO*.LIB xx
- CPPWO*.RSP xx
- CPPWOC3.LIB xx
- CPPWOC3U.MSG xx
- createFloatingFrame
 - IToolBar 622
- createToolBarContainer
 - IToolBar 622
- current
 - IContainerControl::ColumnCursor 236
 - IContainerControl::ObjectCursor 248
 - IContainerControl::TextCursor 253
 - INotebook::Cursor 527
- currentBitmapIndex
 - IAnimatedButton 10
- currentEditColumn
 - IContainerControl 197
- currentEditMLE
 - IContainerControl 197
- currentEditObject
 - IContainerControl 197
- currentGraphicType
 - IGraphicPushButton 442
- currentText
 - ICnrBeginEditEvent 58

- currentText (*continued*)
 - ICnrEndEditEvent 91
 - ICnrReallocStringEvent 117
- Cursor
 - INotebook::Cursor 525
- cursoredChanged
 - ICnrHandler 100
- cursoredObject
 - IContainerControl 212
- ICustomButton 270
 - ICustomButtonDrawEvent 282
- ICustomButton::Style 280
- ICustomButtonDrawEvent 281
- ICustomButtonDrawHandler 285

D

- data
 - IDDEClientConversation 319
 - IDDEDataEvent 332
 - IDDEServerAcknowledgeEvent 350
- data member names, conventions xxi
- dataAsDate
 - IContainerColumn 156
- dataAsIcon
 - IContainerColumn 156
- dataAsNumber
 - IContainerColumn 156
- dataAsString
 - IContainerColumn 157
- dataAsTime
 - IContainerColumn 157
- dataAttributes
 - IContainerColumn 167
- date
 - IContainerColumn 168
- day
 - ICnrDate 62
- DDE4C01E.MSG xx
- IDDEAcknowledgeEvent 289
- IDDEAcknowledgeExecuteEvent 293
- IDDEAcknowledgePokeEvent 296
- IDDEActiveServer 299
- IDDEActiveServerSet 301
- IDDEBeginEvent 303
- IDDEClientAcknowledgeEvent 306
- IDDEClientConversation 309
- IDDEClientEndEvent 323
- IDDEClientHotLinkEvent 326

- IDDEClientHotLinkSet 329
- IDDEDataEvent 331
- IDDEEndEvent 334
- IDDEEvent 337
- IDDEExecuteEvent 340
- IDDEPokeEvent 343
- IDDERequestDataEvent 346
- IDDEServerAcknowledgeEvent 349
- IDDEServerHotLinkEvent 352
- IDDESetAcknowledgeInfoEvent 355
- IDDETopicServer 358
- deckCount
 - ISetCanvas 568
- deckOrientation
 - ISetCanvas 568
- deckOrientationId
 - ISetCanvas 575
- decksByGroup
 - ISetCanvas 576
- decrementUseCount
 - IContainerObject 268
- defaultAttribute
 - IContainerControl 187
- defaultCell
 - IMultiCellCanvas 479
- defaultDataStyle
 - IContainerColumn 157
- defaultGroupPad
 - IToolBar 619
- defaultHeadingStyle
 - IContainerColumn 158
- defaultMargin
 - IToolBar 619
- defaultMisfitWidth
 - IToolBar 619
- defaultPad
 - IToolBar 619
- defaultPushButton
 - ICanvas 28
- defaultStyle
 - IAnimatedButton 11
 - IBitmapControl 21
 - ICanvas 30
 - ICircularSlider 44
 - IContainerControl 220
 - ICustomButton 275
 - IDrawingCanvas 372
 - IFileDialog 383
 - IFontDialog 424
 - IGraphicPushButton 445
- defaultStyle (*continued*)
 - IIconControl 454
 - IMultiCellCanvas 482
 - INotebook 510
 - ISetCanvas 571
 - ISplitCanvas 590
 - IToolBar 615
 - IToolBarButton 638
 - IToolBarContainer 650
 - IViewport 664
- defaultText
 - IFlyOverHelpHandler 411
- defaultTransparentColor
 - IToolBarButton 639
- delayTime
 - IFlyOverHelpHandler 413
- deleteAllObjects
 - IContainerControl 179
- deleteSelectedObjects
 - IContainerControl 179
- deltaReached
 - ICnrHandler 100
- descendentsOf
 - IContainerControl 213
- deselect
 - ICollectionViewComboBox 130
 - ICollectionViewListBox 145
- detailsObjectRectangle
 - IContainerControl 205
- detailsTitleRectangle
 - IContainerControl 192
- detailsView
 - IContainerControl 226
- detailsViewPortOnWindow
 - IContainerControl 210
- detailsViewPortOnWorkspace
 - IContainerControl 210
- detailsViewSplit
 - IContainerControl 193
- detailsViewTitles
 - IContainerControl 226
- detailsViewTitlesId
 - IContainerControl 229
- Direct Manipulation Classes xviii
 - overview xviii
- disableAnimateWhenLatched
 - IAnimatedButton 8
- disableAutoLatch
 - ICustomButton 273
- disableCaching

- disableCaching (*continued*)
 - IContainerControl 199
- disableDataUpdate
 - IContainerColumn 158
 - IContainerControl 216
 - IContainerObject 263
- disableDragDelete
 - IToolBarButton 636
- disableDragDrop
 - IToolBar 612
- disableDragLines
 - IMultiCellCanvas 478
- disableDrawBackground
 - IContainerControl 199
- disableDrawItem
 - IContainerControl 200
- disableDrop
 - IContainerControl 216
 - IContainerObject 263
- disabledText
 - IInfoArea 463
- disabledTextId
 - IInfoArea 468
- disableGridLines
 - IMultiCellCanvas 478
- disableHeadingUpdate
 - IContainerColumn 158
- disableLatching
 - ICustomButton 273
- disableMisfitFiltering
 - IToolBar 613
- disableSizeToGraphic
 - IGraphicPushButton 443
- disableTitleUpdate
 - IContainerControl 181
- dispatchHandlerEvent
 - ICircularSliderNotifyHandler 52
 - ICnrDrawHandler 70
 - ICnrEditHandler 85
 - ICnrHandler 102
 - ICnrMenuHandler 109
 - IContainerControlNotifyHandler 256
 - ICustomButtonDrawHandler 286
 - IDDEClientConversation 317
 - IDDETopicServer 363
 - IFileDialogHandler 402
 - IFlyOverHelpHandler 414
 - IFontDialogHandler 437
 - IInfoArea 466
 - INotebookNotifyHandler 544

- dispatchHandlerEvent (*continued*)
 - IPageHandler 553
- displayValue
 - ICircularSlider 47
- displayWidth
 - IContainerColumn 158
- dragLines
 - IMultiCellCanvas 485
- drawBackground
 - ICnrDrawHandler 69
 - ICustomButtonDrawHandler 286
- drawBorder
 - ICustomButtonDrawHandler 287
- drawButton
 - ICustomButtonDrawHandler 287
- drawDetailsItem
 - ICnrDrawHandler 69
- drawDisabledEmphasis
 - ICustomButtonDrawHandler 287
- drawDown
 - ICustomButtonDrawEvent 282
- drawForeground
 - ICustomButtonDrawHandler 287
- drawIcon
 - ICnrDrawHandler 69
- drawingArea
 - ICustomButtonDrawEvent 282
- IDrawingCanvas 369
- IDrawingCanvas::Style 376
- drawLatched
 - ICustomButtonDrawEvent 283
- drawTab
 - IPageHandler 553
- drawText
 - ICnrDrawHandler 69
- drawTitle
 - ICnrDrawHandler 69
- drawTreeIcon
 - ICnrDrawHandler 69
- drawUp
 - ICustomButtonDrawEvent 283
- Dynamic Data Exchange Classes xviii
 - overview xviii

E

- editColumnTitle
 - IContainerControl 197
- editContainerTitle
 - IContainerControl 197

- editObject
 - IContainerControl 197
- elementAdded
 - ICollectionViewComboBox 127
 - ICollectionViewListBox 143
- elementChanged
 - ICollectionViewComboBox 127
 - ICollectionViewListBox 143
- elementDeleted
 - ICollectionViewComboBox 127
 - ICollectionViewListBox 143
- elementsChanged
 - ICollectionViewComboBox 128
 - ICollectionViewListBox 143
- emHeight
 - IFontDialog 422
- enableAnimateWhenLatched
 - IAnimatedButton 8
- enableAutoLatch
 - ICustomButton 273
- enableCaching
 - IContainerControl 200
- enableDataUpdate
 - IContainerColumn 159
 - IContainerControl 216
 - IContainerObject 263
- enableDragDelete
 - IToolBarButton 636
- enableDragDrop
 - IToolBar 612
- enableDragLines
 - IMultiCellCanvas 478
- enableDrawBackground
 - IContainerControl 200
- enableDrawItem
 - IContainerControl 200
- enableDrop
 - IContainerControl 216
 - IContainerObject 263
- enableExtendedSelect
 - ICollectionViewListBox 146
- enableGridLines
 - IMultiCellCanvas 478
- enableHeadingUpdate
 - IContainerColumn 159
- enableLatching
 - ICustomButton 273
- enableMisfitFiltering
 - IToolBar 613
- enableNotification
 - ICircularSlider 44
 - IContainerControl 204
 - INotebook 498
- enableSizeToGraphic
 - IGraphicPushButton 444
- enableTitleUpdate
 - IContainerControl 181
- end
 - IDDEClientConversation 313
- endAllHotLinks
 - IDDEClientConversation 321
- endConversation
 - IDDETopicServer 360
- endEdit
 - ICnrEditHandler 83
- endHotLink
 - IDDEClientConversation 315
- endHotLinks
 - IDDEClientConversation 315
- enter
 - ICnrHandler 100
- enterId
 - IContainerControl 230
- enterPressed
 - ICnrEnterEvent 94
- executeAcknowledged
 - IDDEClientConversation 320
- executeCommands
 - IDDEClientConversation 315
 - IDDETopicServer 366
- expand
 - IContainerControl 216
 - IToolBar 613
- expandableViewWindow
 - IViewPort 670
- expandForText
 - ISetCanvas 574
- expandTree
 - IContainerControl 220
- extendedSelectChangedId
 - ICollectionViewListBox 150
- extendedSelection
 - IContainerControl 231
- externalLeading
 - IFontDialog 422

F

- FileDialog 378
- FileDialog::Settings 390
- FileDialog::Style 396
- FileDialogEvent 398
- FileDialogHandler 400
- fileLength
 - FileDialogEvent 399
- fileName
 - FileDialog 381
 - FileDialog::Settings 393
 - FileDialogEvent 399
- filter
 - IContainerControl 198
 - FileDialog 386
 - FileDialogHandler 403
 - IToolBar 622
- FilterFn
 - IContainerControl::FilterFn 242
- filterMisfits
 - IToolBar 624
- filterName
 - FileDialogHandler 403
- findTransaction
 - IDDEClientConversation 321
- first
 - ICnrAllocator 55
 - IContainerControl::ColumnCursor 237
 - IContainerControl::ObjectCursor 248
 - IContainerControl::TextCursor 253
 - INotebook::Cursor 527
- firstPage
 - INotebook 502
- fixedWidthOnly
 - IFontDialog 427
- fixupChildren
 - ICanvas 34
- floatingFrame
 - IToolBar 614
- floatingPosition
 - IToolBar 614
- floatingTitle
 - IToolBar 614
- flowedView
 - IContainerControl 226
- flyHelpText
 - IFlyOverHelpHandler 409
- IFlyOverHelpHandler 406

- IFlyText 415
- flyTextControl
 - IFlyOverHelpHandler 410
- flyTextStringTableOffset
 - IFlyOverHelpHandler 412
- IFontDialog 419
- IFontDialog::Settings 430
- IFontDialog::Style 434
- IFontDialogHandler 435
- fontFamily
 - IFontDialog 422
- fontWeight
 - IFontDialog 422
- fontWidth
 - IFontDialog 423
- foregroundColor
 - IContainerControl 207
- format
 - IDDEEvent 338
- FrameCursor
 - IToolBar::FrameCursor 627
- frameToolBarContainer
 - IToolBarContainer 649
- full360
 - ICircularSlider 47
- function arguments, conventions xxii
- function return types, conventions xxi

G

- global names, conventions xxi
- graphicContext
 - ICustomButtonDrawEvent 283
 - IDrawingCanvas 371
- graphicList
 - IDrawingCanvas 372
- IGraphicPushButton 439
- IGraphicPushButton::Style 450
- graphicWindow
 - IGraphicPushButton 442
- gridLines
 - IMultiCellCanvas 485
- groupPad
 - ISetCanvas 569

H

- h files, description xix
- handleAck
 - IDDEClientConversation 318

- handleAck (*continued*)
 - IDDETopicServer 363
- handleAdvise
 - IDDETopicServer 363
- handleCursoredChange
 - IContainerObject 260
- handleData
 - IDDEClientConversation 318
- handleDrawBackground
 - IContainerControl 227
- handleDrawItem
 - IContainerColumn 169
 - IContainerControl 227
- handleDrawTabs
 - INotebook 519
- handleEventsFor
 - ICnrDrawHandler 68
 - ICnrEditHandler 83
 - ICnrHandler 99
 - ICnrMenuHandler 107
 - IFileDialogHandler 401
 - IFlyOverHelpHandler 410
 - IInfoArea 462
- handleExecute
 - IDDETopicServer 363
- handleInitiate
 - IDDETopicServer 363
- handleInitiateAck
 - IDDEClientConversation 318
- handleInuseChange
 - IContainerObject 261
- handleOpen
 - IContainerObject 261
- handlePoke
 - IDDETopicServer 363
- handleRequest
 - IDDETopicServer 364
- handleSelectedChange
 - IContainerObject 261
- handleTerminate
 - IDDEClientConversation 318
 - IDDETopicServer 364
- handleTreeCollapse
 - IContainerObject 261
- handleTreeExpand
 - IContainerObject 261
- handleUnadvise
 - IDDETopicServer 364
- hasDragLines
 - IMultiCellCanvas 478
- hasGridLines
 - IMultiCellCanvas 478
- hasHorizontalSeparator
 - IContainerColumn 159
- hasMixedTargetEmphasis
 - IContainerControl 181
- hasNormalTargetEmphasis
 - IContainerControl 181
- hasOrderedTargetEmphasis
 - IContainerControl 181
- hasTransparentColor
 - IToolBarButton 639
- hasVerticalSeparator
 - IContainerColumn 160
- headingIcon
 - IContainerColumn 160
- headingText
 - IContainerColumn 160
- help
 - ICnrHandler 100
 - IPageHandler 553
- helpButton
 - IFileDialog 387
 - IFontDialog 427
- helpId
 - IContainerColumn 160
 - IContainerObject 262
- helpWindow
 - IPageHelpEvent 557
- hide
 - IContainerColumn 160
 - IContainerObject 264
 - IFlyText 416
- hideDetailsViewTitles
 - IContainerControl 182
- hideObject
 - IContainerControl 216
- hideSeparators
 - IContainerColumn 161
- hideSourceEmphasis
 - IContainerControl 217
- hideSplitBar
 - IContainerControl 193
- hideTitle
 - IContainerControl 182
- hideTitleSeparator
 - IContainerControl 182
- hideTreeLine
 - IContainerControl 220
- hiliteBackgroundColor

- hiliteBackgroundColor (*continued*)
 - INotebook 493
- horizontal
 - ISplitCanvas 593
- horizontalDataAlignment
 - IContainerColumn 161
- horizontalDecks
 - ISetCanvas 576
- horizontalHeadingAlignment
 - IContainerColumn 161
- horizontalScrollBar
 - IViewport 662
- horizontalSeparator
 - IContainerColumn 169
- hotLinkCount
 - IDDEClientConversation 314
 - IDDETopicServer 361
- hotLinkEnded
 - IDDETopicServer 367
- hotLinkInform
 - IDDEClientConversation 320
- hotLinks
 - IDDEClientConversation 314
- hotLinkUpdate
 - IDDETopicServer 360
- hours
 - ICnrTime 123
- hpp files, description xix
- hwndAllocation
 - IContainerControl 225

I

- IAnimatedButton
 - IAnimatedButton 10
- IBitmapControl
 - IBitmapControl 18
- ibmcl.cat xxi
- ICanvas
 - ICanvas 27, 32
- ICircularSlider
 - ICircularSlider 42
- ICircularSliderNotifyHandler
 - ICircularSliderNotifyHandler 51
- ICnrAllocator
 - ICnrAllocator 54
- ICnrBeginEditEvent
 - ICnrBeginEditEvent 57
- ICnrControlList
 - ICnrControlList 60

- ICnrDrawBackgroundEvent
 - ICnrDrawBackgroundEvent 64
- ICnrDrawHandler
 - ICnrDrawHandler 70
- ICnrDrawItemEvent
 - ICnrDrawItemEvent 72
- ICnrEditEvent
 - ICnrEditEvent 77
- ICnrEditHandler
 - ICnrEditHandler 82
- ICnrEmphasisEvent
 - ICnrEmphasisEvent 87
- ICnrEndEditEvent
 - ICnrEndEditEvent 90
- ICnrEnterEvent
 - ICnrEnterEvent 93
- ICnrEvent
 - ICnrEvent 96
- ICnrHandler
 - ICnrHandler 99
- ICnrHelpEvent
 - ICnrHelpEvent 103
- ICnrMenuHandler
 - ICnrMenuHandler 108
- ICnrObjectSet
 - ICnrObjectSet 111
- ICnrQueryDeltaEvent
 - ICnrQueryDeltaEvent 113
- ICnrReallocStringEvent
 - ICnrReallocStringEvent 116
- ICnrScrollEvent
 - ICnrScrollEvent 120
- ICollectionViewComboBox
 - ICollectionViewComboBox 128
- ICollectionViewListBox
 - ICollectionViewListBox 144
- icon
 - IContainerColumn 171
 - IContainerObject 264
 - IGraphicPushButton 443
 - IIconControl 453
 - IIconControl 451
 - IIconControl::Style 459
- iconRectangle
 - IContainerControl 206
- iconSize
 - IContainerControl 182
- IContainerColumn
 - IContainerColumn 155

- IContainerControl
 - IContainerControl 191, 222
- IContainerControlNotifyHandler
 - IContainerControlNotifyHandler 255
- IContainerObject
 - IContainerObject 259
- iconText
 - IContainerObject 264
- iconView
 - IContainerControl 227
- ICustomButton
 - ICustomButton 272, 276
- ICustomButtonDrawEvent
 - ICustomButtonDrawEvent 281
- ICustomButtonDrawHandler
 - ICustomButtonDrawHandler 285
- id
 - IFileDialog 382
 - IFontDialog 423
- IDDEAcknowledgeEvent
 - IDDEAcknowledgeEvent 289
- IDDEAcknowledgeExecuteEvent
 - IDDEAcknowledgeExecuteEvent 293
- IDDEAcknowledgePokeEvent
 - IDDEAcknowledgePokeEvent 296
- IDDEActiveServer
 - IDDEActiveServer 299
- IDDEActiveServerSet
 - IDDEActiveServerSet 301
- IDDEBeginEvent
 - IDDEBeginEvent 303
- IDDEClientAcknowledgeEvent
 - IDDEClientAcknowledgeEvent 306
- IDDEClientConversation
 - IDDEClientConversation 311
- IDDEClientEndEvent
 - IDDEClientEndEvent 323
- IDDEClientHotLinkEvent
 - IDDEClientHotLinkEvent 326
- IDDEClientHotLinkSet
 - IDDEClientHotLinkSet 329
- IDDEDataEvent
 - IDDEDataEvent 331
- IDDEEndEvent
 - IDDEEndEvent 334
- IDDEEvent
 - IDDEEvent 337
- IDDEExecuteEvent
 - IDDEExecuteEvent 340
- IDDEPokeEvent
 - IDDEPokeEvent (continued)
 - IDDEPokeEvent 343
- IDDERequestDataEvent
 - IDDERequestDataEvent 346
- IDDEServerAcknowledgeEvent
 - IDDEServerAcknowledgeEvent 349
- IDDEServerHotLinkEvent
 - IDDEServerHotLinkEvent 352
- IDDESetAcknowledgeInfoEvent
 - IDDESetAcknowledgeInfoEvent 355
- IDDETopicServer
 - IDDETopicServer 359
- IDrawingCanvas
 - IDrawingCanvas 370
- IFileDialog
 - IFileDialog 379
- IFileDialogEvent
 - IFileDialogEvent 398
- IFileDialogHandler
 - IFileDialogHandler 401
- IFlyOverHelpHandler
 - IFlyOverHelpHandler 408
- IFlyText
 - IFlyText 416
- IFontDialog
 - IFontDialog 420
- IFontDialogHandler
 - IFontDialogHandler 436
- IGraphicPushButton
 - IGraphicPushButton 440
- IIconControl
 - IIconControl 452
- IInfoArea
 - IInfoArea 461
- immediateDescendantsOf
 - IContainerControl 213
- IMultiCellCanvas
 - IMultiCellCanvas 476
- inactiveText
 - IInfoArea 463
- inactiveTextId
 - IInfoArea 468
- includeEAS
 - IFileDialog 387
- inConversation
 - IDDEClientConversation 312
- incrementUseCount
 - IContainerObject 268
- IInfoArea
 - IInfoArea 460
- informationFor

informationFor (*continued*)
 InfoArea 466
 initialDelayTime
 IFlyOverHelpHandler 413
 initialDrive
 IFileDialog::Settings 393
 initialFileType
 IFileDialog::Settings 393
 initialize
 ICanvas 31
 IContainerControl 198
 IContainerObject 268
 IToolBar 622
 initialized
 ICnrAllocator 55
 inl files, description xix
 INotebook
 INotebook 496
 INotebookDrawItemEvent
 INotebookDrawItemEvent 540
 INotebookNotifyHandler
 INotebookNotifyHandler 543
 inuseChanged
 ICnrHandler 101
 invalidate
 IContainerColumn 167
 IContainerControl::ColumnCursor 239
 IContainerControl::ObjectCursor 248
 IContainerControl::TextCursor 253
 INotebook::Cursor 526
 IToolBar::FrameCursor 627
 IToolBar::WindowCursor 631
 invisible
 IContainerColumn 169
 IPageEvent
 IPageEvent 546
 IPageHandle
 IPageHandle 549
 IPageHandler
 IPageHandler 552
 IPageHelpEvent
 IPageHelpEvent 556
 IPageRemoveEvent
 IPageRemoveEvent 559
 IPageSelectEvent
 IPageSelectEvent 562
 isAckPositive
 IDDEAcknowledgeEvent 290
 isAckRequested
 IDDEClientHotLinkEvent 327
 isAckRequested (*continued*)
 IDDEDataEvent 332
 isAckToBeginHotLink
 IDDEClientAcknowledgeEvent 307
 isAckToEndHotLink
 IDDEClientAcknowledgeEvent 307
 isAckToHotLinkUpdate
 IDDEServerAcknowledgeEvent 350
 isAckToRequestData
 IDDEClientAcknowledgeEvent 307
 isAcquired
 ICnrEmphasisEvent 88
 isAnimatedWhenLatched
 IAnimatedButton 8
 isAnimationStarted
 IAnimatedButton 8
 isApplicationBusy
 IDDEAcknowledgeEvent 290
 isAttribute
 IContainerObject 268
 isAutoLatchEnabled
 ICustomButton 273
 isAutoSize
 INotebook::PageSettings 530
 isBitmapVisible
 IToolBarButton 640
 isButtonEnabled
 ICustomButtonDrawEvent 283
 isCachingEnabled
 IContainerControl 201
 isCaseSensitive
 IDDEActiveServer 300
 IDDEClientConversation 312
 isCollapsed
 IContainerControl 208
 isColumnExpandable
 IMultiCellCanvas 479
 isColumnRight
 IContainerControl 193
 isCursored
 IContainerControl 208
 isDataFromHotLink
 IDDEDataEvent 332
 isDataString
 ICnrEditHandler 83
 isDataRequested
 IDDEClientHotLinkEvent 327
 IDDEServerHotLinkEvent 353
 isDate
 IContainerColumn 161

- isDefaultTransparentColorSet
 - IToolBarButton 639
- isDetailsData
 - ICnrEditEvent 77
- isDetailsView
 - IContainerControl 194
- isDialogTemplateSet
 - IFileDialog::Settings 391
- isDragStarting
 - IContainerControl 223
- isDrawBackgroundEnabled
 - IContainerControl 201
- isDrawItemEnabled
 - IContainerControl 201
- isDrawTabsEnabled
 - INotebook 511
- isDropOnAble
 - IContainerControl 208
 - IContainerObject 264
- isEmpty
 - INotebook 505
- isEqual
 - IContainerControl::CompareFn 241
- ISetCanvas
 - ISetCanvas 566
- isExpanded
 - IContainerControl 208
 - IToolBar 613
- isExtendedSelection
 - IContainerControl 182
- isFlowed
 - IContainerControl 194
- isFlowedNameView
 - IContainerControl 195
- isFlowedTextView
 - IContainerControl 195
- isHeadingIconHandle
 - IContainerColumn 162
- isHeadingString
 - IContainerColumn 162
- isHeadingWriteable
 - IContainerColumn 162
- isHorizontal
 - ICnrScrollEvent 121
- isIconHandle
 - IContainerColumn 162
- isIconView
 - IContainerControl 195
- isInUse
 - IContainerControl 209

- isInUse (*continued*)
 - IContainerObject 264
- isLatched
 - ICustomButton 273
- isLatchedBackgroundColorHalfTone
 - ICustomButton 271
- isLatchingEnabled
 - ICustomButton 274
- isLeftDetails
 - ICnrEditEvent 77
 - ICnrScrollEvent 121
- isLeftDetailsHeading
 - ICnrEditEvent 78
- isMajorTab
 - INotebook::PageSettings 533
- isMemberOf
 - IContainerControl::FilterFn 243
- isMessageUnderstood
 - IDDEAcknowledgeEvent 290
- isMinorTab
 - INotebook::PageSettings 533
- isMisfitFilteringEnabled
 - IToolBar 614
- isModeless
 - IFileDialog 382
 - IFontDialog 424
- isMoveValid
 - IContainerControl 202
- isMultipleSelection
 - IContainerControl 182
- isNameView
 - IContainerControl 195
- isNumber
 - IContainerColumn 162
- isOpen
 - IContainerObject 264
- isOpenDialog
 - IFileDialog::Settings 394
- isPacingRequested
 - IDDEServerHotLinkEvent 353
- ISplitCanvas
 - ISplitCanvas 587
- isPMCompatible
 - IContainerControl 183
 - INotebook 498
- isPositionSet
 - IFileDialog::Settings 391
- isRefreshOn
 - IContainerControl 211
 - IContainerObject 265

- isRightDetails
 - ICnrEditEvent 78
 - ICnrScrollEvent 121
- isRightDetailsHeading
 - ICnrEditEvent 78
- isRowExpandable
 - IMultiCellCanvas 480
- isSelected
 - IContainerControl 209
- isShowingMiniIcons
 - IContainerControl 195
- isSingleSelection
 - IContainerControl 183
- isSizeToGraphic
 - IGraphicPushButton 444
- isSource
 - IContainerControl 209
- isStandardFormat
 - IToolBarButton 636
- isStatusTextOn
 - INotebook::PageSettings 532
- isString
 - IContainerColumn 162
- isTabStop
 - ICanvas 29
- isTarget
 - IContainerControl 209
- isTextView
 - IContainerControl 196
- isTextVisible
 - IToolBarButton 640
- isTime
 - IContainerColumn 163
- isTitleSeparatorVisible
 - IContainerControl 183
- isTitleVisible
 - IContainerControl 183
- isTitleWindow
 - ICnrEditEvent 78
- isTitleWriteable
 - IContainerControl 183
- isTreeIconView
 - IContainerControl 196
- isTreeNameView
 - IContainerControl 196
- isTreeTextView
 - IContainerControl 196
- isTreeView
 - IContainerControl 196
- IStringGenerator
 - IStringGenerator 598
- IStringGeneratorFn
 - IStringGeneratorFn 602
- IStringGeneratorMemberFn
 - IStringGeneratorMemberFn 605
- IStringGeneratorRefMemberFn
 - IStringGeneratorRefMemberFn 608
- isValid
 - IContainerControl::ColumnCursor 239
 - IContainerControl::ObjectCursor 248
 - IContainerControl::TextCursor 253
 - INotebook::Cursor 526
 - IToolBar::FrameCursor 628
 - IToolBar::WindowCursor 631
- isVertical
 - ICnrScrollEvent 121
- isVisible
 - IContainerColumn 163
 - IContainerControl 207, 209
 - IContainerObject 265
- isWriteable
 - IContainerColumn 163
 - IContainerControl 209
 - IContainerObject 265
- item
 - IDDEEvent 338
- itemChangedId
 - ICollectionViewComboBox 134
 - ICollectionViewListBox 150
- itemId
 - ICnrDrawBackgroundEvent 64
 - INotebookDrawItemEvent 541
- itemPresSpaceHandle
 - ICnrDrawBackgroundEvent 64
 - ICnrDrawItemEvent 73
- itemRect
 - ICnrDrawBackgroundEvent 64
 - ICnrDrawItemEvent 73
- items
 - ICollectionViewComboBox 126
 - ICollectionViewListBox 142
- itemsId
 - ICollectionViewComboBox 134
 - ICollectionViewListBox 150
- itemType
 - ICnrDrawItemEvent 73
- Iterator
 - IContainerControl::Iterator 244

- IToolBar
 - IToolBar 611
- IToolBarButton
 - IToolBarButton 634
- IToolBarContainer
 - IToolBarContainer 649
- IToolBarFrameWindow
 - IToolBarFrameWindow 656
- IViewPort
 - IViewPort 662

J

- jumpToPointer
 - ICircularSlider 47
- justifyData
 - IContainerColumn 163
- justifyHeading
 - IContainerColumn 163

L

- label
 - ICircularSlider 47
- last
 - IContainerControl::ColumnCursor 237
 - IContainerControl::ObjectCursor 248
 - IContainerControl::TextCursor 254
 - INotebook::Cursor 527
- lastPage
 - INotebook 502
- latch
 - IAnimatedButton 8
 - ICustomButton 274
- latchable
 - ICustomButton 278
- latchedBackgroundColor
 - ICustomButton 271
- latchedBitmap
 - IToolBarButton 633
- latchedForegroundColor
 - ICustomButton 271
- latchId
 - ICustomButton 277
- layout
 - ICanvas 34
 - IMultiCellCanvas 484
 - ISetCanvas 574
 - ISplitCanvas 591
 - IViewPort 667

- layoutSize
 - ICanvas 34
- leftAlign
 - ISetCanvas 577
- lib files, description xix
- libbmui.a xxi
- libbmuis.a xxi
- lineCount
 - IInfoArea 462
- lineSpacing
 - IContainerControl 184
- listDelete
 - ICollectionViewConstants 138
- listDeleteAll
 - ICollectionViewConstants 138
- listEnd
 - ICollectionViewConstants 138
- listError
 - ICollectionViewConstants 138
- listInsert
 - ICollectionViewConstants 138
- listMemoryError
 - ICollectionViewConstants 138
- listNone
 - ICollectionViewConstants 139
- location
 - IToolBar 620
 - IToolBarContainer 649
- longHelpText
 - IFlyOverHelpHandler 409
- longStringTableOffset
 - IFlyOverHelpHandler 412
- longTextControl
 - IFlyOverHelpHandler 410

M

- majorTab
 - INotebook::PageSettings 535
- majorTabBackgroundColor
 - INotebook 493
- majorTabBackgroundColorId
 - INotebook 516
- majorTabForegroundColor
 - INotebook 493
- majorTabForegroundColorId
 - INotebook 516
- majorTabsBottom
 - INotebook 519

- majorTabsLeft
 - INotebook 519
- majorTabsRight
 - INotebook 519
- majorTabsTop
 - INotebook 520
- margin
 - ISetCanvas 569
- marginSize
 - IGraphicPushButton 444
- matchForMnemonic
 - ICanvas 29
- member function names, conventions xxi
- menuEnded
 - ICnrMenuHandler 109
 - IInfoArea 467
- menuSelected
 - IInfoArea 467
- midpoint
 - ICircularSlider 47
- miniIcons
 - IContainerControl 227
- minorTab
 - INotebook::PageSettings 535
- minorTabBackgroundColor
 - INotebook 493
- minorTabBackgroundColorId
 - INotebook 516
- minorTabForegroundColor
 - INotebook 493
- minorTabForegroundColorId
 - INotebook 517
- minutes
 - ICnrTime 123
- missingText
 - IInfoArea 464
- missingTextId
 - IInfoArea 468
- mixedTargetEmphasis
 - IContainerControl 228
- mleHandler
 - ICnrEditHandler 84
- modeless
 - IFileDialog 387
 - IFontDialog 428
- modelessFileApplied
 - IFileDialogHandler 403
- modelessFontApplied
 - IFontDialogHandler 437

- modelessResults
 - IFileDialogHandler 403
 - IFontDialogHandler 437
- month
 - ICnrDate 62
- moveAfter
 - IToolBar 617
- moveBefore
 - IToolBar 618
- moveIconTo
 - IContainerControl 206
- moveObjectTo
 - IContainerControl 203
- moveSizeTo
 - IBitmapControl 21
 - IGraphicPushButton 445
 - IIconControl 455
 - ISetCanvas 572
- moveToFirst
 - IToolBar 618
- moveToLast
 - IToolBar 618
- IMultiCellCanvas 470
- IMultiCellCanvas::Style 487
- multiLineEdit
 - ICnrEditHandler 84
- Multimedia Classes xviii
 - overview xviii
- multipleSelection
 - IContainerControl 231
- multiSelection
 - IFileDialog 387

N

- nameView
 - IContainerControl 228
- newText
 - ICnrEndEditEvent 91
 - ICnrReallocStringEvent 117
- newTextSize
 - ICnrReallocStringEvent 117
- next
 - ICnrAllocator 55
 - IContainerControl::ColumnCursor 237
 - IContainerControl::ObjectCursor 249
 - IContainerControl::TextCursor 254
 - INotebook::Cursor 527
- nextAvailable
 - ICnrAllocator 55

- nextPage
 - INotebook 503
- nlsCompare
 - IContainerControl 219
- noAttribute
 - INotebook::PageSettings 535
- noDragDelete
 - IToolBarButton 643
- noDragDrop
 - IToolBar 624
- noHorizontalScrollBar
 - IViewPort 671
- nominalPointSize
 - IFontDialog 423
- noSelection
 - ICollectionViewComboBox 134
 - ICollectionViewListBox 151
- noSharedObjects
 - IContainerControl 231
- noSplitBars
 - ISplitCanvas 593
- noStyle
 - IFileDialog 387
 - IFontDialog 428
- noSynthesize
 - IFontDialog 428
- INotebook 489
 - IPageEvent 547
 - IPageHelpEvent 557
 - IPageRemoveEvent 560
 - IPageSelectEvent 563
- INotebook::Cursor 525
- INotebook::PageSettings 529
- INotebook::PageSettings::Attribute 537
- INotebook::Style 539
- INotebookDrawItemEvent 540
- INotebookNotifyHandler 543
- notebookSize
 - INotebook 505
- noTicks
 - ICircularSlider 47
- noVerticalScrollBar
 - IViewPort 671
- noViewWindowFill
 - IViewPort 671
- number
 - IContainerColumn 169
- numberOfColumnChanges
 - IContainerControl 201
- numberOfObjectChanges

- numberOfObjectChanges (*continued*)
 - IContainerControl 201
- numerations, conventions xxi

O

- object
 - ICnrDrawItemEvent 73
 - ICnrEditEvent 79
 - ICnrEmphasisEvent 88
 - ICnrEnterEvent 94
 - ICnrHelpEvent 104
- objectAt
 - IContainerControl 213
- objectCopy
 - IContainerObject 262
- objectCount
 - IContainerControl 213
- ObjectCursor
 - IContainerControl::ObjectCursor 246
- objectList
 - IContainerControl 213
- objectUnderPoint
 - IContainerControl 206
- ok
 - IFileDialog 386
 - IFontDialog 427
- okButtonText
 - IFileDialog::Settings 391
- operator =
 - INotebook::PageSettings 530
 - IStringGenerator 598
- operator ==
 - IContainerControl 191
 - IContainerObject 262
- operator delete
 - IContainerObject 262
- operator new
 - IContainerObject 262
- operator Value
 - IPageHandle 550
- orderedTargetEmphasis
 - IContainerControl 228
- orientation
 - INotebook 498
 - ISplitCanvas 589
- orientationId
 - INotebook 517
 - ISplitCanvas 592

- origDefaultButtonHandle
 - ICanvas 29
- outstandingTransactionCount
 - IDDEClientConversation 312

P

- packEven
 - ISetCanvas 577
- packExpanded
 - ISetCanvas 577
- packTight
 - ISetCanvas 577
- packType
 - ISetCanvas 569
- pad
 - ISetCanvas 570
- page
 - INotebookDrawItemEvent 541
- pageBackgroundColor
 - INotebook 494
- pageBackgroundColorId
 - INotebook 517
- IPageEvent 546
- pageHandle 549
 - IPageEvent 547
 - IPageHelpEvent 558
 - IPageSelectEvent 563
- IPageHandler 551
- IPageHelpEvent 556
- IPageRemoveEvent 559
- IPageSelectEvent 562
- pageSettings
 - INotebook 502
 - INotebook::PageSettings 531
- pageSize
 - INotebook 505
- pagesToEnd
 - INotebook 506
- pagesToMajorTab
 - INotebook 506
- pagesToMinorTab
 - INotebook 506
- pageWindow
 - IPageRemoveEvent 560
- parentObject
 - IContainerControl 213
- passEventToOwner
 - ICanvas 32

- pmCompatible
 - IContainerControl 232
 - INotebook 520
- pointSize
 - IFontDialog 423
- pokeAcknowledged
 - IDDEClientConversation 320
- pokeData
 - IDDEClientConversation 316
 - IDDETopicServer 367
- pokedData
 - IDDEAcknowledgePokeEvent 297
 - IDDEPokeEvent 344
- polygonTabs
 - INotebook 520
- popupMenuObject
 - ICnrMenuHandler 107
- position
 - IFileDialog::Settings 391
 - ISetCanvas 572
- positionViewWindow
 - IViewPort 667
- preload
 - IFileDialog 387
- Presentation Manager messages
- pressedOK
 - IFileDialog 382
 - IFontDialog 424
- previous
 - IContainerControl::ColumnCursor 237
 - IContainerControl::ObjectCursor 249
 - IContainerControl::TextCursor 254
 - INotebook::Cursor 527
- previousPage
 - INotebook 503
- previousSelectedPageHandle
 - IPageSelectEvent 564
- proportionalOnly
 - IFontDialog 428
- proportionalTicks
 - ICircularSlider 48

R

- radius
 - ICircularSlider 40
- readOnly
 - IContainerColumn 169
 - IContainerControl 232

- readOnlyHeading
 - IContainerColumn 170
- readOnlyTitle
 - IContainerControl 228
- reallocateString
 - ICnrEditHandler 84
 - ICnrReallocStringEvent 117
- reallocateText
 - ICnrReallocStringEvent 117
- refresh
 - IContainerControl 211
 - IContainerObject 265
- refreshAllContainers
 - IContainerControl 212
- refreshTabs
 - INotebook 511
- registerCallbacks
 - ICanvas 33
 - IContainerControl 223
 - IFileDialog 385
 - INotebook 515
- relativeWindowRect
 - IFlyText 417
- remove
 - IPageHandler 554
 - IToolBar 618
- removeAllObjects
 - IContainerControl 180
- removeAllPages
 - INotebook 504
- removeColumn
 - IContainerControl 193
- removeColumnAt
 - IContainerControl 193
- removeFromCell
 - IMultiCellCanvas 475
- removeHelpText
 - IFlyOverHelpHandler 409
- removeId
 - IContainerControl 230
- removeInUse
 - IContainerControl 217
 - IContainerObject 265
- removeLink
 - IDDETopicServer 364
- removeObject
 - IContainerControl 180
- removeObjectAt
 - IContainerControl 180
- removePage
 - INotebook 504
- removeSelected
 - IContainerControl 217
- removeSelectedObjects
 - IContainerControl 180
- removeSourceEmphasis
 - ICnrMenuHandler 109
- removeTabSection
 - INotebook 504
- requestAck
 - IDDERequestDataEvent 347
- requestData
 - IDDEClientConversation 316
 - IDDETopicServer 367
- requestHotLinkData
 - IDDETopicServer 368
- resetBackgroundColor
 - IContainerControl 207
- resetButton
 - IFontDialog 428
- resetForegroundColor
 - IContainerControl 207
- resetLatchedBackgroundColor
 - ICustomButton 271
- resetLatchedForegroundColor
 - ICustomButton 271
- resetMajorTabBackgroundColor
 - INotebook 494
- resetMajorTabForegroundColor
 - INotebook 494
- resetMinorTabBackgroundColor
 - INotebook 494
- resetMinorTabForegroundColor
 - INotebook 494
- resetPageBackgroundColor
 - INotebook 495
- resetSplitBarEdgeColor
 - ISplitCanvas 586
- resetSplitBarMiddleColor
 - ISplitCanvas 586
- resetTransparentColor
 - IToolBarButton 639
- resize
 - IPageHandler 554
- resourceLibrary
 - IFlyOverHelpHandler 411
 - IInfoArea 463
- resourceLibraryId
 - IInfoArea 468

- returnValue
 - IFileDialog 382
 - IFontDialog 424
- rightAlign
 - ISetCanvas 577
- rotationIncrement
 - ICircularSlider 40
- roundedTabs
 - INotebook 520
- rowHeight
 - IMultiCellCanvas 480

S

- sample directory location xxiv
- saveAsEAType
 - IFileDialog 381
- scroll
 - IContainerControl 214
- scrollDetailsHorizontally
 - IContainerControl 214
- scrollHorizontally
 - IContainerControl 215
- scrollToObject
 - IContainerControl 215
- scrollVertically
 - IContainerControl 215
- scrollViewHorizontallyTo
 - IViewPort 665
- scrollViewVerticallyTo
 - IViewPort 665
- seconds
 - ICnrTime 123
- select
 - ICollectionViewComboBox 130
 - ICollectionViewListBox 146
 - IPageHandler 554
- selectableListbox
 - IFileDialog 388
- selectedChanged
 - ICnrHandler 101
- selectedCollectionPosition
 - ICollectionViewComboBox 130
 - ICollectionViewListBox 146
- selectedElement
 - ICollectionViewComboBox 130
 - ICollectionViewListBox 147
- selectedElements
 - ICollectionViewListBox 147

- selectedFileCount
 - IFileDialog 381
- selectId
 - IContainerControl 230
- selectPending
 - IPageHandler 554
- sendEvent
 - IContainerControl 207
- serverHandle
 - IDDETopicServer 361
- setAlignment
 - ISetCanvas 570
- setAnimationRate
 - IAnimatedButton 9
- setApplicationBusy
 - IDDESetAcknowledgeInfoEvent 356
- setApplicationSpecificData
 - IDDESetAcknowledgeInfoEvent 356
- setArmRange
 - ICircularSlider 40
- setAttributes
 - IContainerControl 225
 - IContainerObject 268
- setBackgroundColor
 - IContainerControl 208
 - INotebook 495
- setBase
 - IContainerObject 268
- setBinding
 - INotebook 492
- setBitmap
 - IBitmapControl 18
 - IToolBarButton 633
- setBitmaps
 - IAnimatedButton 10
- setBuffer
 - IDDEEvent 339
- setButtonView
 - IToolBar 616
- ISetCanvas 565
- ISetCanvas::Style 581
- setCaseSensitive
 - IDDEBeginEvent 304
- setClosed
 - IContainerObject 265
- setColumnInfo
 - IContainerColumn 167
- setColumnWidth
 - IMultiCellCanvas 480

- setContainer
 - IContainerColumn 167
- setContainerAttributes
 - IContainerControl 225
- setCurrent
 - IContainerControl::ColumnCursor 237
 - IContainerControl::ObjectCursor 249
 - IContainerControl::TextCursor 254
 - INotebook::Cursor 527
- setCurrentBitmapIndex
 - IAnimatedButton 10
- setCursor
 - IContainerControl 217
- setData
 - IDDERequestDataEvent 347
- setDataAttributes
 - IContainerColumn 168
- setDataOffset
 - IContainerColumn 164
- setDeckCount
 - ISetCanvas 568
- setDeckOrientation
 - ISetCanvas 568
- setDecrementBitmaps
 - ICircularSlider 40
- setDefaultAttribute
 - IContainerControl 187
- setDefaultCell
 - IMultiCellCanvas 481
- setDefaultDataStyle
 - IContainerColumn 157
- setDefaultGroupPad
 - IToolBar 619
- setDefaultHeadingStyle
 - IContainerColumn 158
- setDefaultMargin
 - IToolBar 619
- setDefaultMisfitWidth
 - IToolBar 620
- setDefaultPad
 - IToolBar 620
- setDefaultStyle
 - IAnimatedButton 12
 - IBitmapControl 21
 - ICanvas 30
 - ICircularSlider 44
 - IContainerControl 220
 - ICustomButton 275
 - IDrawingCanvas 373
 - IFileDialog 383
- setDefaultStyle (*continued*)
 - IFontDialog 425
 - IGraphicPushButton 445
 - IIconControl 455
 - IMultiCellCanvas 482
 - INotebook 510
 - ISetCanvas 571
 - ISplitCanvas 590
 - IToolBar 616
 - IToolBarButton 638
 - IToolBarContainer 650
 - IViewPort 665
- setDefaultText
 - IFlyOverHelpHandler 411
- setDefaultTransparentColor
 - IToolBarButton 640
- setDelayTime
 - IFlyOverHelpHandler 413
- setDeleteColumnsOnClose
 - IContainerControl 187
- setDeleteObjectsOnClose
 - IContainerControl 188
- setDetailsViewSplit
 - IContainerControl 194
- setDialogTemplate
 - IFileDialog::Settings 392
 - IFontDialog::Settings 431
- setDisabledText
 - IInfoArea 464
- setDisplayPS
 - IFontDialog::Settings 432
- setDisplayWidth
 - IContainerColumn 164
- setDrawingArea
 - ICustomButtonDrawEvent 283
- setEditColumn
 - IContainerControl 198
- setEditMLE
 - IContainerControl 198
- setEditObject
 - IContainerControl 198
- setEmphasis
 - IContainerControl 225
 - IContainerObject 268
- setExtendedSelection
 - IContainerControl 184
- setFamily
 - IFontDialog::Settings 432
- setFileName
 - IFileDialog::Settings 394

- setFloatingPosition
 - IToolBar 614
- setFloatingTitle
 - IToolBar 615
- setFlyTextControl
 - IFlyOverHelpHandler 410
- setFlyTextStringTableOffset
 - IFlyOverHelpHandler 412
- setFont
 - ICanvas 29
 - IFontDialog::Settings 432
- setForegroundColor
 - IContainerControl 208
 - INotebook 495
- setGraphic
 - IGraphicPushButton 443
- setGraphicContext
 - IDrawingCanvas 371
- setGraphicList
 - IDrawingCanvas 372
- setGroupPad
 - ISetCanvas 570
- setHeadingIcon
 - IContainerColumn 164
- setHeadingText
 - IContainerColumn 165
- setHelpId
 - IContainerColumn 165
- setHelpText
 - IFlyOverHelpHandler 409
- setIcon
 - IContainerObject 266
 - IIconControl 454
- setIconSize
 - IContainerControl 184
- setIconText
 - IContainerObject 266
- setId
 - IFileDialog 383
 - IFontDialog 424
- setInactiveText
 - IInfoArea 464
- setIncrementBitmaps
 - ICircularSlider 41
- setInitialDelayTime
 - IFlyOverHelpHandler 413
- setInitialDrive
 - IFileDialog::Settings 394
- setInitialFileType
 - IFileDialog::Settings 394
- setInUse
 - IContainerControl 218
 - IContainerObject 266
- setItems
 - ICollectionViewComboBox 126
 - ICollectionViewListBox 142
- setLatchedBackgroundColor
 - ICustomButton 272
- setLatchedBitmap
 - IToolBarButton 634
- setLatchedForegroundColor
 - ICustomButton 272
- setLayoutDistorted
 - IBitmapControl 20
 - ICanvas 29
 - IMultiCellCanvas 479
 - ISetCanvas 569
 - ISplitCanvas 588
 - IToolBar 611
 - IToolBarButton 634
 - IToolBarContainer 648
 - IViewPort 664
- setLayoutSize
 - ICanvas 34
- setLineCount
 - IInfoArea 462
- setLineSpacing
 - IContainerControl 184
- setLocation
 - IToolBar 620
 - IToolBarContainer 649
- setLongStringTableOffset
 - IFlyOverHelpHandler 412
- setLongTextControl
 - IFlyOverHelpHandler 410
- setMajorTabBackgroundColor
 - INotebook 495
- setMajorTabForegroundColor
 - INotebook 495
- setMajorTabSize
 - INotebook 511
- setMargin
 - ISetCanvas 570
- setMarginSize
 - IGraphicPushButton 444
- setMessageNotUnderstood
 - IDDESetAcknowledgeInfoEvent 356
- setMinorTabBackgroundColor
 - INotebook 496

setMinorTabForegroundColor
 INotebook 496
 setMinorTabSize
 INotebook 512
 setMissingText
 IInfoArea 464
 setMixedTargetEmphasis
 IContainerControl 184
 setMLEHandler
 ICnrEditHandler 84
 setMultipleSelection
 IContainerControl 185
 setNormalTargetEmphasis
 IContainerControl 185
 setOKButtonText
 IFileDialog::Settings 392
 setOpen
 IContainerObject 266
 setOpenDialog
 IFileDialog::Settings 394
 setOrderedTargetEmphasis
 IContainerControl 185
 setOrientation
 INotebook 499
 ISplitCanvas 589
 setPackType
 ISetCanvas 570
 setPad
 ISetCanvas 571
 setPageBackgroundColor
 INotebook 496
 setPageButtonSize
 INotebook 501
 setPointSize
 IFontDialog::Settings 432
 setPosition
 IFileDialog::Settings 392
 IFontDialog::Settings 431
 setPreviewText
 IFontDialog::Settings 431
 setPrinterPS
 IFontDialog::Settings 433
 setRefreshOff
 IContainerControl 212
 IContainerObject 267
 setRefreshOn
 IContainerControl 212
 IContainerObject 267
 setRelativeWindowRect
 IFlyText 417
 setResourceLibrary
 IFlyOverHelpHandler 411
 IInfoArea 463
 setRotationIncrement
 ICircularSlider 41
 setRowHeight
 IMultiCellCanvas 481
 setSaveAsDialog
 IFileDialog::Settings 395
 setSelected
 IContainerControl 218
 setSingleSelection
 IContainerControl 185
 setSizeList
 IFontDialog::Settings 432
 setSplitBarEdgeColor
 ISplitCanvas 586
 setSplitBarMiddleColor
 ISplitCanvas 586
 setSplitBarThickness
 ISplitCanvas 589
 setSplitWindowPercentage
 ISplitCanvas 585
 setStandardBitmapSize
 IToolBarButton 637
 setStandardTextLines
 IToolBarButton 637
 setStandardTextWidth
 IToolBarButton 637
 setStatus
 IDDEEvent 339
 setStatusText
 INotebook 508
 INotebook::PageSettings 533
 setStatusTextAlignment
 INotebook 509
 setStringGenerator
 ICollectionViewComboBox 131
 ICollectionViewListBox 148
 setStringTableOffset
 IInfoArea 465
 setTabBitmap
 INotebook 512
 INotebook::PageSettings 534
 setTabShape
 INotebook 513
 setTabText
 INotebook 513
 INotebook::PageSettings 534
 setTabTextAlignment

setTabTextAlignment (<i>continued</i>)	
INotebook	514
setText	
IFlyText	416
ISetCanvas	572
setTickSpacing	
ICircularSlider	41
Settings	
IFileDialog::Settings	391
IFontDialog::Settings	431
setTitle	
IContainerControl	185
IFileDialog::Settings	392
IFontDialog::Settings	431
setTitleAlignment	
IContainerControl	186
setTitleAttributes	
IContainerColumn	168
setTitleText	
IToolBarFrameWindow	656
setToFirst	
IContainerControl::ColumnCursor	238
IContainerControl::ObjectCursor	247
IContainerControl::TextCursor	252
INotebook::Cursor	526
IToolBar::FrameCursor	628
IToolBar::WindowCursor	631
setToLast	
IContainerControl::ColumnCursor	238
IContainerControl::ObjectCursor	247
IContainerControl::TextCursor	252
INotebook::Cursor	526
setToNext	
IContainerControl::ColumnCursor	238
IContainerControl::ObjectCursor	247
IContainerControl::TextCursor	252
INotebook::Cursor	526
IToolBar::FrameCursor	628
IToolBar::WindowCursor	631
setTopLeftViewPoint	
IViewPort	668
setToPrevious	
IContainerControl::ColumnCursor	238
IContainerControl::ObjectCursor	247
IContainerControl::TextCursor	253
INotebook::Cursor	526
setTransparentColor	
IToolBarButton	640
setTreeExpandIconSize	
IContainerControl	220
setTreeItemIcons	
IContainerControl	221
setTreeViewIndent	
IContainerControl	221
setupScrollBars	
IViewPort	668
setUserData	
ICustomButton	275
INotebook	492
INotebook::PageSettings	530
setValue	
ICircularSlider	41
setView	
IToolBarButton	640
setViewWindowSize	
IViewPort	665
setWindow	
INotebook	507
show	
IContainerColumn	165
IContainerObject	267
IFileDialog	384
showDetailsView	
IContainerControl	188
showDetailsViewTitles	
IContainerControl	186
showFlowedNameView	
IContainerControl	189
showFlowedTextView	
IContainerControl	189
showIconView	
IContainerControl	189
showMiniIcons	
IContainerControl	189
showNameView	
IContainerControl	189
showObject	
IContainerControl	218
showSeparators	
IContainerColumn	166
showSourceEmphasis	
IContainerControl	218
showSplitBar	
IContainerControl	194
showTextView	
IContainerControl	190
showTitle	
IContainerControl	186
showTitleSeparator	
IContainerControl	186

- showTreeIconView
 - IContainerControl 190
- showTreeLine
 - IContainerControl 221
- showTreeNameView
 - IContainerControl 190
- showTreeTextView
 - IContainerControl 190
- singleSelection
 - IContainerControl 232
- size
 - ISetCanvas 573
- sizeToBitmap
 - IBitmapControl 23
- sizeToGraphic
 - IGraphicPushButton 447
- sizeToIcon
 - IIconControl 457
- solidBinding
 - INotebook 520
- sort
 - IContainerControl 219
- sortByIconText
 - IContainerControl 219
- sourceOfEnd
 - IDDEEndEvent 335
- spiralBinding
 - INotebook 520
- splitBarEdgeColor
 - ISplitCanvas 587
- splitBarMiddleColor
 - ISplitCanvas 587
- splitBarOffset
 - IContainerControl 194
- splitBarThickness
 - ISplitCanvas 590
- ISplitCanvas 583
- ISplitCanvas::Style 595
- splitWindowPercentage
 - ISplitCanvas 586
- squareTabs
 - INotebook 520
- Standard Control Classes xviii
 - overview xviii
- standardBitmapSize
 - IToolBarButton 637
- standardFormat
 - IToolBarButton 644
- standardTextLines
 - IToolBarButton 637
- standardTextWidth
 - IToolBarButton 637
- startAnimation
 - IAnimatedButton 9
- status
 - IDDEEvent 339
- statusText
 - INotebook::PageSettings 533
- statusTextAlignment
 - INotebook 509
- statusTextCenter
 - INotebook 520
- statusTextLeft
 - INotebook 521
- statusTextOn
 - INotebook::PageSettings 535
- statusTextRight
 - INotebook 521
- stopAnimation
 - IAnimatedButton 9
- stopHandlingEventsFor
 - ICnrDrawHandler 68
 - ICnrEditHandler 83
 - ICnrHandler 99
 - ICnrMenuHandler 107
 - IFileDialogHandler 402
 - IFlyOverHelpHandler 410
 - IInfoArea 462
- string
 - IContainerColumn 172
- stringFor
 - IStringGenerator 599
 - IStringGeneratorFn 602
 - IStringGeneratorMemberFn 605
 - IStringGeneratorRefMemberFn 608
- stringGenerator 597
 - ICollectionViewComboBox 131
 - ICollectionViewListBox 148
- IStringGeneratorFn 601
- IStringGeneratorMemberFn 604
- IStringGeneratorRefMemberFn 607
- stringTableOffset
 - IInfoArea 465
- supportedTopics
 - IDDEClientConversation 310
- supportingApplications
 - IDDEClientConversation 310

T

- tabBitmap
 - INotebook::PageSettings 534
 - IPageRemoveEvent 560
- tabShape
 - INotebook 514
- tabText
 - INotebook::PageSettings 534
- tabTextAlignment
 - INotebook 514
- tabTextCenter
 - INotebook 521
- tabTextLeft
 - INotebook 521
- tabTextRight
 - INotebook 521
- text
 - ISetCanvas 572
- TextCursor
 - IContainerControl::TextCursor 251
- textId
 - ISetCanvas 575
- textRectangle
 - IContainerControl 206
- textRef
 - ICnrEditEvent 79
- textSize
 - ICnrEditEvent 80
- textView
 - IContainerControl 228
- textVisible
 - IToolBarButton 644
- tickSpacing
 - ICircularSlider 41
- time
 - IContainerColumn 170
- title
 - IContainerControl 186
 - IFileDialog::Settings 393
- titleAttributes
 - IContainerColumn 168
- titleId
 - IContainerControl 230
- titleRectangle
 - IContainerControl 187
- titleSeparator
 - IContainerControl 229
- titleText
 - IToolBarFrameWindow 657
- titleVisibleId
 - IContainerControl 230
- IToolBar 610
 - IToolBar::FrameCursor 627
 - IToolBar::Style 629
 - IToolBar::WindowCursor 630
- toolBarAt
 - IToolBar 612
- IToolBarButton 632
 - IToolBarButton::Style 647
- toolBarContainer 648
 - IToolBar 616
- IToolBarContainer::Style 654
- IToolBarFrameWindow 655
- topAlign
 - ISetCanvas 578
- topHandle
 - IContainerControl 222
 - ISetCanvas 568
- topic
 - IDDEActiveServer 300
 - IDDEBeginEvent 304
 - IDDEClientConversation 312
 - IDDEClientEndEvent 324
 - IDDETopicServer 362
- topLeftViewPoint
 - IViewport 668
- topPage
 - INotebook 503
- totalPages
 - INotebook 507
- trackId
 - ICircularSlider 46
- transactionType
 - IDDEAcknowledgeEvent 291
- transparentColor
 - IToolBarButton 640
- treeCollapsed
 - ICnrHandler 101
- treeExpanded
 - ICnrHandler 101
- treeView
 - IContainerControl 229
- turnToPage
 - INotebook 503
- Two-Dimensional Graphic Classes xviii
 - overview xviii
- type names, conventions xxi

U

- ucReserved
 - ICnrTime 123
- unlatch
 - ICustomButton 274
- unregisterCallbacks
 - ICanvas 33
 - IContainerControl 223
 - IFileDialog 385
 - INotebook 515
- updateForInsert
 - ICnrAllocator 56
- useCount
 - IContainerObject 269
- useDefaultPaintHandler
 - IDrawingCanvas 375
- useIdForBitmap
 - IToolBarButton 644
- useIdForText
 - IToolBarButton 644
- userData
 - ICustomButton 275
 - INotebook::PageSettings 530

V

- validate
 - IFileDialogHandler 404
- validateName
 - IFileDialogHandler 404
- validObject
 - ICnrEnterEvent 94
- value
 - ICircularSlider 42
- valueId
 - ICircularSlider 46
- vectorOnly
 - IFontDialog 428
- verifyPointers
 - IContainerControl 232
- vertical
 - ISplitCanvas 593
- verticalDataAlignment
 - IContainerColumn 166
- verticalDecks
 - ISetCanvas 578
- verticalHeadingAlignment
 - IContainerColumn 166
- verticalScrollBar

- verticalScrollBar (*continued*)

- IViewport 662
- verticalSeparator
 - IContainerColumn 170
- view
 - IToolBarButton 641
- IViewport 660
- IViewport::Style 673
- viewPortOnWindow
 - IContainerControl 210
- viewPortOnWorkspace
 - IContainerControl 210
- viewWindow
 - IViewport 666
- viewWindowDrawRectangle
 - IViewport 666
- viewWindowSize
 - IViewport 666
- viewWindowSizeId
 - IViewport 669
- visibleRectangle
 - IIconControl 451
 - ISplitCanvas 589
- visibleTitle
 - IContainerControl 229
- visibleTreeLine
 - IContainerControl 229

W

- willDeleteColumnsOnClose
 - IContainerControl 188
- willDeleteObjectsOnClose
 - IContainerControl 188
- window
 - INotebook 508
- windowAt
 - IToolBar 612
- WindowCursor
 - IToolBar::WindowCursor 630
- windowInCell
 - IMultiCellCanvas 476
- windowScrolled
 - ICnrHandler 101

X

- X/Motif messages
- xHeight
 - IFontDialog 423

xxxxxxx.inf xxi

Y
year
ICnrDate 62

Communicating Your Comments to IBM

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume III

Version 3.5

Publication No. S33H-5041-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
 - United States and Canada: 416-448-6161
 - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
 - Internet: torrcf@vnet.ibm.com
 - IBMLink: [toribm\(torrcf\)](mailto:toribm(torrcf)@vnet.ibm.com)
 - IBM/PROFS: [torolab4\(torrcf\)](mailto:torolab4(torrcf)@vnet.ibm.com)
 - IBMMAIL: [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@vnet.ibm.com)

Readers' Comments — We'd Like to Hear from You

IBM VisualAge for C++ for Windows
Open Class Library Reference
Volume III

Version 3.5

Publication No. S33H-5041-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
S33H-5041-00

IBM®

Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape

Please do not staple

Fold and Tape

S33H-5041-00

Cut or Fold
Along Line

IBM®

Part Number: 33H5041

Printed in U.S.A.

S33H-5041-00



33H5041

